MSX
FAGOT-BARRALY
PROGRAMAÇÃO
EM ASSEMBLER
MSX
MSX
MSX
EDITORA
MANOLE
LTDA.

PROGRAMAÇÃO
EM ASSEMBLER

PROGRAMAÇÃO EM ASSEMBLER

GEORGES FAGOT-BARRALY

Editora Manole Ltda.
1988

ÍNDICE

INTRODUÇÃO

Este livro é destinado a todos que conhecem os elementos básicos da programação BASIC do MSX e que desejam agora atingir a etapa seguinte, a linguagem de máquina e sua forma mais compreensível: o assembler.

Há obras muito bem feitas que tratam deste assunto, mas a maioria delas dirige-se a leitores já iniciados, desprezando aqueles que dão os primeiros passos neste campo. Por isso julgamos que, no mundo dos livros, havia lugar para uma obra que daria a mão aos principiantes e guiaria, com mil cuidados, a sua entrada no universo fascinante dos microprocessadores.

Em todo o decorrer deste estudo, fixamos a seguinte regra: para cada um dos exemplos analisados, daremos a parte assembler, a sua tradução em linguagem de máquina e a maneira de incluir estes códigos de máquina num programa BASIC. Diante do seu computador, por conseguinte, o leitor poderá pôr em prática os conhecimentos que acabou de adquirir.

O Capítulo 1 dará as noções indispensáveis da aritmética binária pois, é preciso não esquecer, um computador não conhece, de fato, senão os algarismos 0 e 1.

O Capitulo 2 lembrará como é concebida a memória da tela do MSX. Este estudo tornou-se necessário tendo-se em vista que a maioria dos programas escritos em linguagem de máquina são animações de tipo vídeo. Acrescentamos que isto nos permitirá, após comprovação, apreciar os resultados de uma boa parte dos programas assembler deste livro. Encontra-se também neste capítulo um programa de demonstração que nos fará ver a diferença flagrante nas velocidades de execução de um programa BASIC e do seu equivalente assembler.

O Capítulo 3 nos fará penetrar no interior do microprocessador; é um capítulo dedicado aos diferentes registradores, cujo conhecimento é obrigatório para abordar a continuação deste livro. Encontraremos também neste capítulo o estudo das diferentes maneiras de se utilizar uma instrução assembler conforme o modo de endereçamento escolhido.

O Capítulo 4 conterá o estudo do nosso primeiro programa escrito em assembler: os menores detalhes serão explicados. Algumas

páginas serão reservadas aos leitores que podem dispor do editor/assembler Zen.

O Capítulo 5 analisará as principais instruções necessárias à programação do microprocessador do MSX. Numerosos exemplos serão fornecidos e sempre com a maneira como o BASIC e a linguagem de máquina serão ligados um ao outro. Encadeamos o estudo das diversas instruções sem preocupação de qualquer ordem lógica ou alfabética: somente a noção de progressão nos guiou.

Nossa principal aspiração é ter feito um livro facilmente acessível, um livro que não se fecha ao término de algumas páginas diante da suposição de uma excessiva dificuldade.

1

A ARITMÉTICA BINÁRIA

OS SISTEMAS DE NUMERAÇÃO

A base dez

O sistema de numeração de base 10 é o sistema que utilizamos na vida diária. É conhecido pelo nome de sistema decimal e nele o número 10 tem papel primordial.

Para começar o nosso estudo, lembraremos o que valem as potências de 10:

$$10^0 = 1$$
$$10^1 = 10$$
$$10^2 = 10 \times 10 = 100$$
$$10^3 = 10 \times 10 \times 10 = 1000$$

Por convenção, todo número com expoente 0 é igual a 1, e 10 não foge à regra: $10^0 = 1$.

Com o auxílio destas potências, é possivel escrever qualquer número inteiro.

$$2548 = 2000 + 500 + 40 + 8$$

ou $2000 = 2 \times 1000 = 2 \times 10^3$

$500 = 5 \times 100 = 5 \times 10^2$

$40 = 4 \times 10 = 4 \times 10^1$

$8 = 8 \times 1 = 8 \times 10^0$

o que dá: $2548 = 2 \times 10^3 + 5 \times 10^2 + 4 \times 10^1 + 8 \times 10^0$.

De maneira análoga, teremos:

$$4706 = 4000 + 700 + 6$$

ou seja: $4706 = 4 \times 10^3 + 7 \times 10^2 + 0 \times 10^1 + 6 \times 10^0$.

$10^3 = 1000$	$10^2 = 100$	$10^1 = 10$	$10^0 = 1$
2	5	4	8
4	7	0	6

Naturalmente, temos a possibilidade de escolher números maiores bastando tomar potências de 10 com um expoente superior.

Em tudo isso não há nada complicado. Passaremos ao estudo de uma outra base mas, primeiramente, observemos algo que encontrataremos em todo este capítulo: os algarismos utilizados na base 10 vão de 0 a 9; são todos inferiores a esta base.

A base cinco

As potências de 5 calculam-se facilmente: $5^0 = 1$; $5^1 = 5$; $5^2 = 25$; $5^3 = 125$. Para escrever um número na base 5, é preciso constituir uma tabela análoga à precedente mas, é claro, a sua primeira linha será escrita com as potências de 5. Tomemos por exemplo traduzir 138 no sistema de base 5:

$5^3 = 125$	$5^2 = 25$	$5^1 = 5$	$5^0 = 1$
1	0	2	3

Procuramos quantos múltiplos de $125(5^3)$ são contidos em 138:

1 vez e sobra 13 : $138 = 1 \times 125 + 13$.

Depois procuramos quantas vezes $25(5^2)$ é contido em 13:

Zero vezes e continua a sobrar 13 : $138 = 1 \times 125 + 0 \times 25 + 13$.

Precisamos então procurar quantas vezes $5(5^1)$ está contido em 13:

2 vezes e sobra 3 : $138 = 1 \times 125 + 0 \times 25 + 2 \times 5 + 3$.

Última fase da operação: no resto que agora vale 3, quantas vezes está contido $1(5^0)$?

3 vezes e não sobra nada: $138 = 1 \times 125 + 0 \times 25 + 2 \times 5 + 3 \times 1$.

Logo, temos em resumo:

$$138 = 1 \times 5^3 + 0 \times 5^2 + 2 \times 5^1 + 3 \times 5^0$$

Assim deduzimos que 138 se escreve 1023 na base 5.

Tomaremos um segundo exemplo: qual é o valor de 279 na base 5?

$5^3 = 125$	$5^2 = 25$	$5^1 = 5$	$5^0 = 1$
2	1	0	4

$$279 = 2 \times 125 + 1 \times 25 + 0 \times 5 + 4 \times 1$$

ou $279 = 2 \times 5^3 + 1 \times 5^2 + 0 \times 5^1 + 4 \times 5^0$

Por conseguinte, 279 se escreve 2104 na base 5.

Na prática, para escrever um número decimal em outra base, utilizamos geralmente o método "das divisões sucessivas"

```
279 | 5
  4 |___
    | 55 | 5
    |  0 |____
         | 11 | 5
         |  1 |___
              | 2 | 5
              | 2 | 0
```

Este método consiste em dividir o número por 5, depois o quociente por 5, depois o novo quociente obtido por 5 e assim até que o último quociente seja nulo. Em seguida, basta escrever a lista dos diferentes restos tomando o cuidado essencial de copiá-los na ordem inversa. No nosso exemplo, sendo os restos 4,0,1,2 escreve-se então: 279 = 2104 (base 5).

Se temos que traduzir em decimal um número já escrito na base 5, é preciso inscrever este número em uma tabela concebida como as precedentes e calculá-lo em seguida.

Vamos escrever 3421 (base 5) na base 10.

$5^3 = 125$	$5^2 = 25$	$5^1 = 5$	$5^0 = 1$
3	4	2	1

Deduz-se que 3421 (base 5) $= 3 \times 5^3 + 4 \times 5^2 + 2 \times 5^1 + 1 \times 5^0$.
E obtém-se: 3421 (base 5) $= 3 \times 125 + 4 \times 25 + 2 \times 5 + 1 \times 1 =$
$= 486$.

Observemos, para terminar, que os únicos algarismos utilizados na base 5 são 0,1,2,3,4.

Aconselhamos ao leitor que faça alguns exercícios, tomando números aleatórios até certificar-se de que tudo o que foi visto está bem assimilado. Não porque a base 5 tenha uma determinada importância em informática, mas porque ela permite compreender sem dificuldades os mecanismos dos sistemas de numeração.

A base dois

Chegamos agora ao centro do problema: o sistema de numeração (chamado sistema binário) utilizado pelos computadores.

Primeiramente, as potências de 2 : $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$.

Agora, um exemplo: vamos escrever 23 em binário:

$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
1	0	1	1	1

A maior potência de dois que cabe em 23 é $16(2^4)$: o resto é 7. Pode-se em seguida fazer caber $8(2^3)$ em 7?. A resposta é **não** e o algarismo 0 foi colocado na casa correspondente.

Por outro lado $4(2^2)$ está contido em 7: escreve-se o algarismo 1 na terceira casa e observa-se o novo resto: 3.

Sendo $2(2^1)$ menor que 3, escreve-se o algarismo 1 na quarta casa e uma vez que o resto então vale 1, precisamos ainda escrever 1, mas desta vez na última coluna.

23 = 10111 (base 2)

Felizmente, para nós, o método das divisões sucessivas por 2 vai nos dar a resposta de uma maneira mais segura e rápida:

```
23 | 2
 1 |11 | 2
   | 1 | 5 | 2
       | 1 | 2 | 2
           | 0 | 1 | 2
               | 1 | 0
```

23 = 10111 (2)

Seguem outros exemplos cujos cálculos intermediários serão deixados por conta do leitor:

34 = 100010 (2)
150 = 10010110 (2)
255 = 11111111 (2)

Falta ver como passar da base 2 à base 10.

Admitamos que se queira escrever 1111011 em decimal. Reconstitui-se a tabela na qual são indicadas as potências de 2 e escreve-se o nosso número:

$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
1	1	1	1	0	1	1

Gasta-se mais tempo para fazer a tabela do que para obter a resposta.

$$1111011 = 64 + 32 + 16 + 8 + 2 + 1 = 123 \text{ (decimal)}$$

É necessário observar que, no que acabamos de ver, os únicos algarismos utilizados são o 0 e o 1, isto é, os algarismos inferiores à base.

A base dezesseis

É o sistema (chamado hexadecimal ou hexa) do qual os profissionais de processamento de dados não podem prescindir, embora à primeira vista possamos nos perguntar que relação tem o seu estudo com o presente livro.

Os 16 algarismos necessários à escrita nesta base são primeiramente 0,1,2,3,4,5,6,7,8,9...

Mas depois de 9, já o 10 não pode ser utilizado pois é escrito com 2 algarismos. Os 6 algarismos que faltam foram substituídos pelas primeiras letras do alfabeto.

Algarismos	A	B	C	D	E	F
Valores	10	11	12	13	14	15

Assim 12 se escreve como C, 14 como E.

Aqui também vão se aplicar os métodos de conversão estudados nos parágrafos precedentes.

Por exemplo, para escrever 300 na base 16: as divisões sucessivas devem se fazer por 16.

```
300 | 16
  C | 18 | 16
    |  2 |  1 | 16
         |  1 | 0
```

300 = 12C (hexa)

Passemos a outro exemplo depois de ter observado que o resto da primeira divisão, que valia 12, foi substituído por C.

```
5032 | 16
   8 | 314 | 16
     |   A | 19 | 16
     |     |  3 |  1 | 16
     |     |    |  1 | 0
```

5032 = 13A8 (hexa)

Se desejamos traduzir em decimal um número já escrito em base 16, utilizamos as potências de 16.

$16^0 = 1 \quad 16^1 = 16 \quad 16^2 = 256 \quad 16^3 = 4096$

3D4F (hexa) escreve-se $3 \times 16^3 + D \times 16^2 + 4 \times 16^1 + F \times 16^0$,
logo $3D4F = 3 \times 4096 + 13 \times 256 + 4 \times 16 + 15$,
ou seja 3D4F = 15695 (base decimal).

Será necessário lembrar aos usuários do MSX que existe uma função BASIC, HEX$, que dá imediatamente o valor hexadecimal de um número decimal.

PRINT HEX$ (15695) e aparece na tela do computador 3D4F

Precisamos agora compreender qual o interesse da informática pelo sistema hexadecimal e para isso comparar as representações de um número decimal, quando se quer escrevê-lo na base 2 ou na base 16.

183 (decimal) = 1011 0111 (binário)
183 (decimal) = B 7 (hexa)

Separamos os oito algarismos binários em dois grupos de quatro:

1011 e 0111

ou $1011 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8+2+1 = 11$ (decimal)

e $0111 = 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 4+2+1 = 7$ (decimal)

Observando que 11 decimal se escreve B em hexadecimal, vê-se imediatamente a correspondência entre as bases 2 e 16. É perfeitamente possível passar diretamente da base 2 à base 16 sem ter que conhecer precisamente o número decimal de que se trata.

Experimentemos agora partindo do número decimal 143 que se escreve 10001111 na base 2:

$\underbrace{1000}_{8}\ \underbrace{1111}_{F}$ = 8F em hexadecimal

É evidente que se passará também facilmente da base 16 à base 2 tendo bem clara a tabela seguinte:

Decimal	Binário	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Qual será, por exemplo, em binário o número decimal 4A?

Resposta: $\underbrace{0100}_{4}\ \underbrace{1010}_{A}$

E o número hexadecimal 37E?

Resposta: $\underbrace{0011}_{3}\ \underbrace{0111}_{7}\ \underbrace{1110}_{E}$

Neste último exemplo, os dois primeiros algarismos 0 são inúteis e servem somente para a compreensão da regra que será preciso respeitar sempre: a repartição do número binário deve-se fazer por grupos de quatro e isto sempre partindo da direita.

Uma vez que o MSX dispõe da função HEX$, é possivel evitar a tarefa fastidiosa de conversão de um número binário, graças à utilização intermediária da base 16.

Admitamos que se queira converter 1000 (decimal) em binário:

PRINT HEX$ (1000) dá 3E8

Deduz-se sem dificuldade o resultado procurado:

```
0011  1110  1000
 3     E     8
```

Esta resposta poderá naturalmente ser verificada utilizando-se a função BIN$.

OPERAÇÕES NAS BASES 2 E 16

Vamos nos limitar neste parágrafo à adição e à subtração.

Adição

Considera-se o mecanismo da adição no nosso sistema decimal e aplica-se à base 2.

```
   207
 + 321
 -----
 = 528
```

Neste exemplo, os algarismos de cada coluna se somam; como 10 nunca é atingido (a base, o que não se pode esquecer), não há nenhum problema. O mesmo ocorre nas adições binárias seguintes, pois os totais nunca ultrapassam 2 (valor da base binária):

```
   101100        100011
 + 010001      + 000100
 --------      --------
 = 111101      = 100111
```

Falta ver o caso da reserva:

```
    1
   447
 + 223
 -----
 = 670
```

Nesta adição decimal, 7 e 3 dão 10, isto é, exatamente o valor da base. Escreve-se então 0 embaixo dos algarismos 7 e 3, depois transporta-se 1 para a coluna seguinte. Procederemos do mesmo modo com o sistema binário.

```
     1
  10001
+ 01001
-------
= 11010
```

A soma dos dois algarismos da direita dá 2 (valor da base). O último algarismo do resultado será portanto um 0 e a reserva 1 será escrita em cima da coluna seguinte. O resto dos cálculos se efetua em seguida, sem dificuldade.

Treinemos ainda:

```
     1 1
  100101
+ 000101
--------
= 101010
```

Continua tudo certo; passemos a outro exemplo:

```
      11
  101011
+ 010011
--------
= 111110
```

Como a soma dos dois algarismos da direita deu 2, escreveu-se 0 como último algarismo para a resposta e reservou-se 1. A adição desta reserva com os dois algarismos 1 da segunda coluna dá 3, o que se traduz escrevendo-se o algarismo 1 na resposta e colocando-se uma reserva em cima da terceira coluna.

É preciso reconhecer que o risco de erro não é pequeno quando se efetuam cálculos em binário. Por isso um método freqüentemente utilizado consiste em traduzir os números em hexadecimal, somá-los então, em seguida reconverter, se necessário, o resultado na base 2.

Vamos fazer em hexadecimal as seguintes adições:

```
                      1
  34B5             5264
+ 6614             A32E
------           ------
= 9AC9           = F592
```

Lembrando a correspondência:

A - 10 B = 11 C = 12 D = 13 E = 14 F = 15

compreende-se imediatamente como a primeira operação foi feita.

$$\begin{aligned}&5 + 4 = 9\\&B + 1 = 11 + 1 - C\\&4 + 6 = 10 = A\\&3 + 6 = 9\end{aligned}$$

Quanto à segunda adição, as coisas se decompõem da seguinte maneira:

$$4 + E = 4 + 14 = 18.$$

A reserva que corresponde a 10 no nosso sistema habitual é igual a 16 no sistema hexadecimal. O que faz com que depois de ter colocado a reserva em cima da segunda coluna, sobrará 2 que será escrito como algarismo da direita da resposta que se completa em seguida por:

$$\begin{aligned}&1 + 6 + 2 = 9\\&2 + 3 = 5\\&5 + A = 5 + 10 = 15 = F\end{aligned}$$

Outros exemplos:

$$\begin{array}{r}1\ \ 1\\4BC3\\+\ 2A2F\\\hline=\ 75F2\end{array}\qquad\qquad\begin{array}{r}111\\FFFF\\+\ FFFF\\\hline=\ 1FFFE\end{array}$$

Estamos de acordo, não é? No sistema hexadecimal só há reserva a partir de 16.

Subtração

Guardemos o sistema precedente de numeração e interessemo-nos pelo cálculo de uma diferença:

$$\begin{array}{r}9AE7\\-\ 49B3\\\hline=\ 5134\end{array}$$

É, sem dúvida, muito fácil de compreender:

7 − 3 = 4
E − B = 14 − 11 = 3
A − 9 = 10 − 9 = 1
9 − 4 = 5

Agora, vejamos as reservas:

```
  9B54
- 6A29
------
= 312B
```

Tem-se tendência a dizer 14 menos 9, pois a força do hábito nos leva a acrescentar uma dezena a 4. Na realidade, uma vez que estamos em hexadecimal, não é dez que se deve acrescentar a 4, mas dezesseis. Trata-se, por conseguinte, 20 menos 9: sobra 11, isto é B. Naturalmente a reserva não deve ser perdida na continuação dos cálculos.

5 − 3 (do qual 1 de reserva) = 2
B − A = 11 − 10 = 1
9 − 6 = 3

Outros dois exemplos:

```
  4A85          ABCD
- 1F2E        - 2FFF
------        ------
= 2B57        = 7BCE
```

Os usuários do MSX terão todas as facilidades para se familiarizarem com este gênero de exercício. Para se verificarem pela máquina estes dois cálculos, bastará digitar:

PRINT HEX$ (&H4A85 − &H1F2E)

e

PRINT HEX$ (&HABCD − &H2FFF)

Chegamos agora ao cálculo da diferença entre dois números escritos no sistema binário. O método de subtração direta pode ser empregado:

```
  101011
- 001001
--------
= 100010
```

Os programadores preferem a este um outro método, o chamado "complemento da base dois", pois compreende-se muito bem que a subtração que acaba de ser efetuada teria sido mais complicada se tivesse havido reservas.

O complemento da base dois

Consideremos o número decimal 17.

A sua conversão em binário dá 10001. Para obter o complemento de 2 deste número, respeitam-se as três etapas seguintes:

- escreve-se o nosso número com oito algarismos acrescentando-se à esquerda tantos zeros quanto necessário:
  00010001
- substitui-se cada 0 por 1 e cada 1 por 0:
  11101110
- soma-se 00000001 a este resultado:
  11101111

O número obtido é chamado de complemento da base 2 (com oito algarismos) de 17 e o computador considerará que é o oposto de 17, isto é, o número −17. É isto mesmo, você viu muito bem, no modo complemento da base 2, o número binário 11101111 é igual a −17!

Como tirar a prova? Partindo da simples idéia que consiste em dizer: uma vez que somando 17 e o seu oposto −17, obtém-se 0, deve-se, normalmente somando 00010001 e 11101111, obter também 0.

Vamos ver isto:

```
    11111111
     00010001
+    11101111
-------------
= (1)00000000
```

Os dois algarismos 1 da direita fazem aparecer uma reserva que encontramos em seguida de coluna em coluna. No entanto, é preciso observar que não se deve levar em conta a última reserva e que adotaremos o hábito de negligenciá-la. Veremos em breve que o computador também procede assim: para ele igualmente a última reserva da esquerda será desprezada.

Outro exemplo: tentemos escrever −50 em binário na forma de complemento da base 2.

00110010	50 decimal
11001101	algarismos invertidos
11001110	acréscimo de 1

logo −50 escreve-se 11001110 em binário
ou C E em hexadecimal

Eis exatamente alguns resultados que devem permitir ao leitor assimilar perfeitamente o modo como o computador escreve os números negativos;

−5 (decimal) = 11111011 (binário) = FB (hexa)
−20 (decimal) = 11101100 (binário) = EC (hexa)
−100 (decimal) = 10011100 (binário) = 9C (hexa)

Antes de passar para outro assunto, voltemos um instante ao modo com que se deve agir para fazer uma diferença binária, agora que sabemos utilizar a técnica do complemento da base 2.
Por exemplo, vamos calcular 101000 − 10111.
Procura-se o oposto do segundo termo da subtração em modo complemento da base dois: obtém-se 11101001.
Falta então somar o primeiro termo com o oposto do segundo:

```
    111 1
    00101000
+   11101001
-------------
= (1)00010001
```

A resposta é a seguinte: 101000 − 10111 = 10001.

OPERAÇÕES LÓGICAS

Além dos cálculos aritméticos habituais, podemos com os números binários efetuar operações de um tipo especial chamadas de operações lógicas. Elas não apresentam nenhuma dificuldade pois em nenhum caso ocorre o problema das reservas.

O OU lógico

Esta operação obedece às seguintes regras:

0	0	1	1
OU 0	OU 1	OU 0	OU 1
= 0	= 1	= 1	= 1

É a mesma coisa com números binários maiores:

```
   101101              101000
OU 110101           OU 001100
=  111101           =  101100
```

O MSX dispõe de uma instrução que efetua este tipo de cálculos: é a palavra chave OR. Peçamos-lhe alguns resultados:

PRINT 46 OR 100 ; resposta : 110

```
   101110 <———— 46
OR 1100100 <———— 100
=  1101110 <———— 110
```

PRINT 50 OR 0 ; resposta : 50

```
   110010 <———— 50
OR 000000 <———— 0
=  110010 <———— 50
```

O operador OR vai nos ser útil em assembler pois permitirá forçar um dos algarismos binários a passar a 1. Vejamos como:

PRINT 82 OR 1 ; resposta : 83

```
   1010010 <———— 82
OR 0000001 <———— 1
=  1010011 <———— 83
```

PRINT 91 OR 1 ; resposta : 91

```
   1011011 <———— 91
OR 0000001 <———— 1
=  1011011 <———— 91
```

No primeiro exemplo, parte-se de um número cujo último algarismo binário (bit O) é igual a O. Após a utilização de OR 1, este último algarismo passou a 1 sem que nenhum dos outros algarismos tenha sido modificado.

No segundo caso, começamos com um número que já terminava em 1. OR 1 não modificou nem este número nem naturalmente nenhum outro. Conclui-se por conseguinte que se efetuarmos OR 1 com qualquer número, teremos um resultado cujo último algarismo (bit 0) valerá obrigatoriamente 1.

De modo análogo, calculando OR 4 com qualquer número, estaremos certos que o terceiro algarismo partindo da direita é um 1 (bit 2):

PRINT 19 OR 4 ; resposta : **23**

```
     10011 _______ 19
  OR 00100 _______ 4
  = 10111 _______ 23
```

O terceiro algarismo passou realmente a 1.

PRINT 52 OR 4 ; resposta : **52**

```
     110100 _______ 52
  OR 000100 _______ 4
  = 110100 _______ 52
```

O terceiro algarismo permaneceu 1.

O E lógico

O E lógico é definido pelas seguintes regras:

```
     0        0        1        1
   E 0      E 1      E 0      E 1
  ----     ----     ----     ----
   = 0      = 0      = 0      = 1
```

Alguns exemplos:

```
     101100          101000
   E 011001        E 110111
  ---------       ---------
  = 001000        = 100000
```

Podemos fazer com que o computador realize estes cálculos e desta vez é a palavra reservada AND que vai nos servir.

PRINT 30 AND 40 ; resposta : 8

```
         11110 ________ 30
AND     101000 ________ 40
=       001000 ________ 8
```

Encontra-se a instrução AND em assembler pois, graças a ela, podemos levar a 0 qualquer algarismo binário. Suponhamos que temos um número e que queremos forçar a 0 o seu algarismo da direita (bit 0). Utilizaremos AND 254 e vejamos porque:

PRINT 201 AND 254 ; resposta : 200

```
       11001001 ________ 201
AND    11111110 ________ 254
=      11001000 ________ 200
```

Só o último algarismo foi posto em 0, os outros permaneceram os mesmos. Com efeito, 254 tem a particularidade de ser constituído de sete algarismos 1 seguidos de um único 0.

Se tivéssemos partido de um número já terminado em 0, AND 254 não teria modificado nada, o que nos permite concluir o seguinte: qualquer que seja o número considerado, combinando-o com 254 podemos estar certos que ele terminará em 0.

É possível anular qualquer algarismo de um número com o operador AND. Por exemplo, AND 124 anulará o algarismo da esquerda (bit 7) mas ao mesmo tempo os dois algarismos da direita (bits 0 e 1) de qualquer número de oito algarismos.

PRINT 245 AND 124 ; resposta : 116

```
       11110101 ________ 245
AND    01111100 ________ 124
=      01110100 ________ 116
```

O OU exclusivo lógico

O OU exclusivo (notação XOR) obedece às mesmas regras que o OU já definido, exceto para a quarta parte:

0	0	1	1
XOR 0	XOR 1	XOR 0	XOR 1
= 0	= 1	= 1	= 0

O resultado só é igual a 1 quando um e somente um dos algarismos for igual a 1.

Vamos digitar:

PRINT 30 XOR 40 ; resposta : 54

```
      11110 ________ 30
XOR 101000 ________ 40
=   110110 ________ 54
```

PRINT 25 XOR 100 ; resposta : 125

```
        11001 ________ 25
XOR 1100100 ________ 100
=     1111101 ________ 125
```

O operador XOR é acionado cada vez que se quer fazer mudar para 1 os algarismos 0 e para 0 os algarismos 1. Suponhamos que se queira mudar de posição o último algarismo (bit 0) de um dado número: combinaremos o algarismo com XOR 1. Se o número terminava em 0, ele terminará então em 1, mas ao contrário, se o seu último algarismo era 1, será automaticamente 0. Treinando:

PRINT 28 XOR 1 ; resposta : 29

```
      11100 ________ 28
XOR 00001 ________ 1
=     11101 ________ 29
```

PRINT 31 XOR 1 ; resposta : 30

```
      11111 ________ 31
XOR 00001 ________ 1
=     11110 ________ 30
```

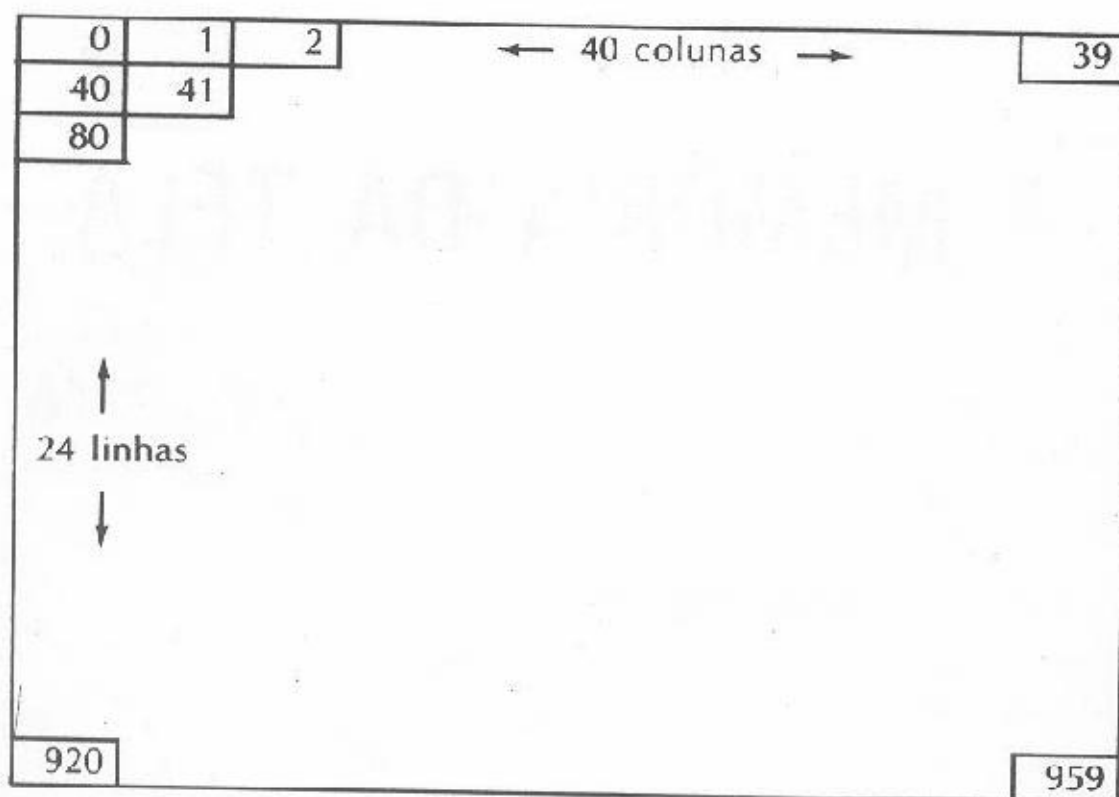
Evidentemente, XOR pode ser utilizado para alterar de um estado para outro qualquer dos algarismos sem modificar os outros. Por exemplo, XOR 5 só mudará os estados do primeiro e do terceiro algarismos partindo da direita (5 é igual a 101 em binário).

2

A MEMÓRIA DA TELA

O MSX transmite ao televisor uma imagem cujas características são dadas pela instrução SCREEN. Quatro tipos de resolução são permitidos e a cada um deles é associado um método bem determinado para obtenção de cores. Estes quatro modos não podem coabitar no computador, pois todos eles utilizam o mesmo dispositivo de memória para guardar a cópia da imagem que aparece na tela.

MODO TEXTO 40 COLUNAS (SCREEN 0)

Memória da tela (960 bytes)

A tela contém 40 caracteres de largura e 24 de altura. O que quer dizer que o computador dispõe de 960 caracteres (40 * 24). Estes caracteres são numerados de 0 (no alto à esquerda) a 959 (embaixo à direita). A cada um destes caracteres corresponde um byte da memória da tela. No Capítulo 3 teremos oportunidade de ver em de-

talhe o que é um byte, mas por enquanto fiquemos cientes que um byte é perfeitamente comparável a uma pequena gaveta na qual se guarda um valor compreendido entre 0 e 255.

O MSX BASIC dispõe de um comando especial para modificar os bytes do vídeo. Vamos utilizá-lo:

```
VPOKE 0, 65
```

A letra A maiúscula aparece no alto e à esquerda da tela. Isto se explica pelo fato de se ter guardado o número 65 na gaveta nº 0. Ora, qual é o código da letra A? 65 não é. Por isso mesmo foi ela, e não outra letra, que apareceu na tela.

Um segundo exemplo:

```
VPOKE 39, 49
```

Desta vez, guarda-se o número 49 na gaveta nº. 39. Diz-se que se escreve o valor 49 no byte cujo endereço é 39. O resultado da operação é claramente visível: o algarismo 1 (código ASCII 49) desenhou-se no alto da tela, à direita.

Continuemos:

```
VPOKE 500, 1
```

500 é o endereço de um byte que corresponde ao centro da imagem. A escrita do algarismo 1 neste byte faz aparecer a cabeça de um homenzinho divertido.

Você está vendo, portanto, que com VPOKE pode-se fazer aparecer todos os caracteres habituais — letras maiúsculas, minúsculas, algarismos etc. e também uma série de pequenos desenhos. Não deixe de tornar visíveis os 256 gráficos possíveis digitando a seguinte linha:

```
FOR I = 0 TO 255 : VPOKE I , I : NEXT
```

Se você compreendeu como funcionava VPOKE, você não terá dificuldade para compreender para que serve a instrução gêmea, VPEEK. Digite esta frase:

```
WIDTH ( 40 ) : CLS : LOCATE 3 , 0 : PRINT "MSX"
```

Depois, pergunte ao computador:

```
PRINT VPEEK (3); resposta = 77 (código ASCII de M)
PRINT VPEEK (4); resposta = 83 (código ASCII de S)
PRINT VPEEK (5); resposta = 88 (código ASCII de X)
```

VPEEK (3) deu-nos o conteúdo do byte (da gaveta se você preferir) nº 3 que é realmente o código da letra M que encontramos. O mesmo ocorreu com os bytes 4 e 5, VPEEK nos trouxe os seus conteúdos.

Tabela dos caracteres (2 048 bytes)

Números dos bytes	Caracteres
2048-2049.....2055	Caractere nº 0
2056-2057.....2063	Caractere nº 1
2064-2065.....2071	Caractere nº 2
...	
...	
...	
2432-2433.....2439	algarismo 0 : ASCII 48 (2432=2048+48*8)
2440-2441.....2447	algarismo 1 : ASCII 49 (2440=2048+49*8)
...	
...	
...	
2568-2569.....2575	Letra A : ASCII 65 (2568=2048+65*8)
2576-2577.....2583	Letra B : ASCII 66 (2576=2048+66*8)
...	
...	
...	
2824-2825.....2831	Letra a : ASCII 97 (2824=2048+97*8)
...	
...	
...	
4088-4089.....4095	Caractere nº 255

Se os 256 caracteres gráficos padrão do MSX não lhe bastam, damos aqui o meio de definir outros:

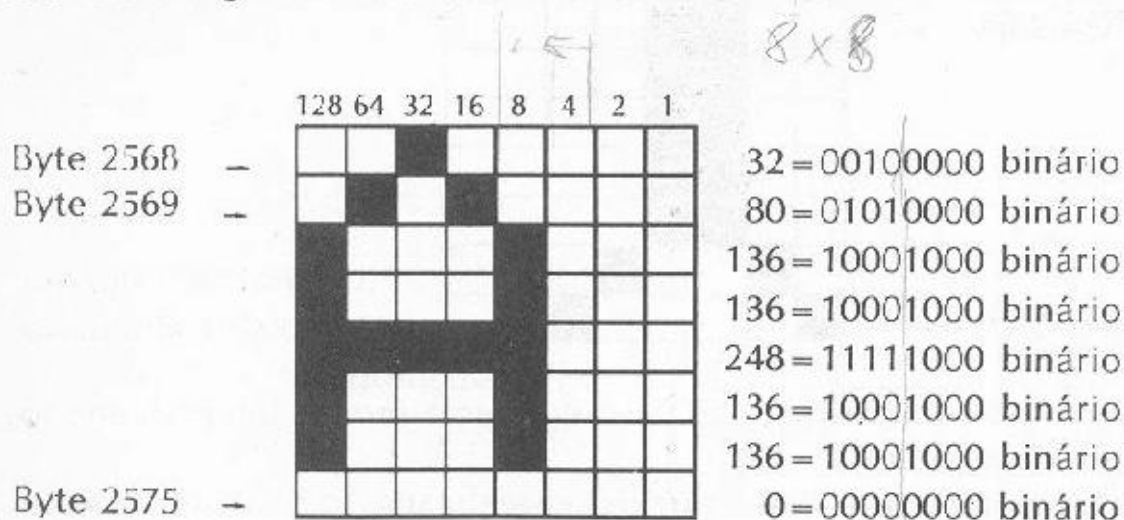
```
FOR I = 2568 TO 2575 : PRINT VPEEK ( I ) ; : NEXT
```

As respostas dadas pela máquina são as seguintes:

```
32  80  136  136  248  136  136  0
```

Se nos esforçarmos para traduzir estes números em binário, obteremos o seguinte desenho:

128 64 32 16 8 4 2 1

Byte 2568 –	32 = 00100000 binário
Byte 2569 –	80 = 01010000 binário
	136 = 10001000 binário
	136 = 10001000 binário
	248 = 11111000 binário
	136 = 10001000 binário
	136 = 10001000 binário
Byte 2575 –	0 = 00000000 binário

Não há possibilidade de dúvida, a série de 8 bytes corresponde ao caractere A maiúsculo. Isto explica, por conseguinte, como o computador encontra o desenho desta letra quando deseja fazê-lo aparecer em modo 0.

Não hesite em fazer outros testes:

```
      FOR I = 2432 TO 2439 : PRINT VPEEK ( I ) ; : NEXT
ou    FOR I = 2824 TO 2831 : PRINT VPEEK ( I ) ; : NEXT
```

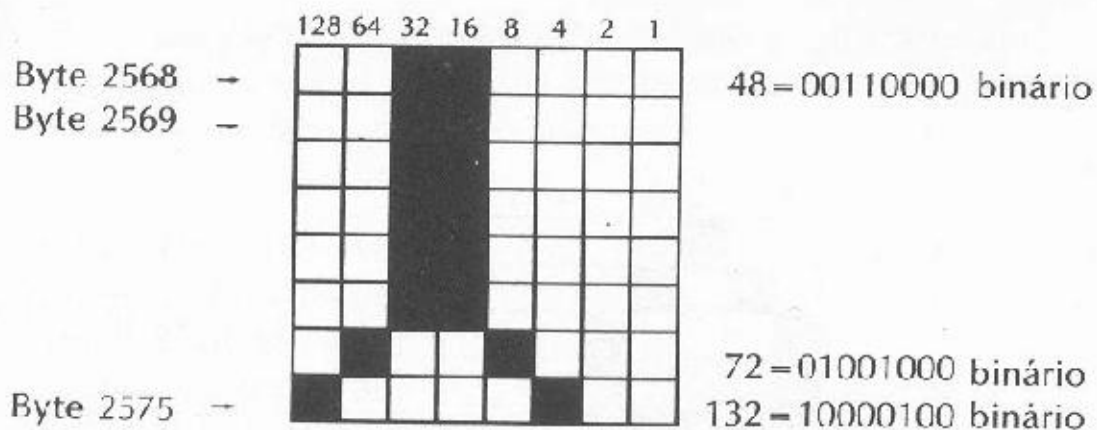
Você achará de um lado o algarismo 0 e do outro a letra minúscula a.

Observemos que o desenho de cada caractere precisa de 8 bytes e que em cada um destes bytes somente os 6 algarismos da esquerda são utilizados; os dois últimos valem sempre 0.

Em resumo, os grafismos dos 256 caracteres padrão são inscritos em uma tabela de 2 048 bytes (8 bytes * 256 caracteres) e o primeiro byte desta tabela tem por endereço 2 048.

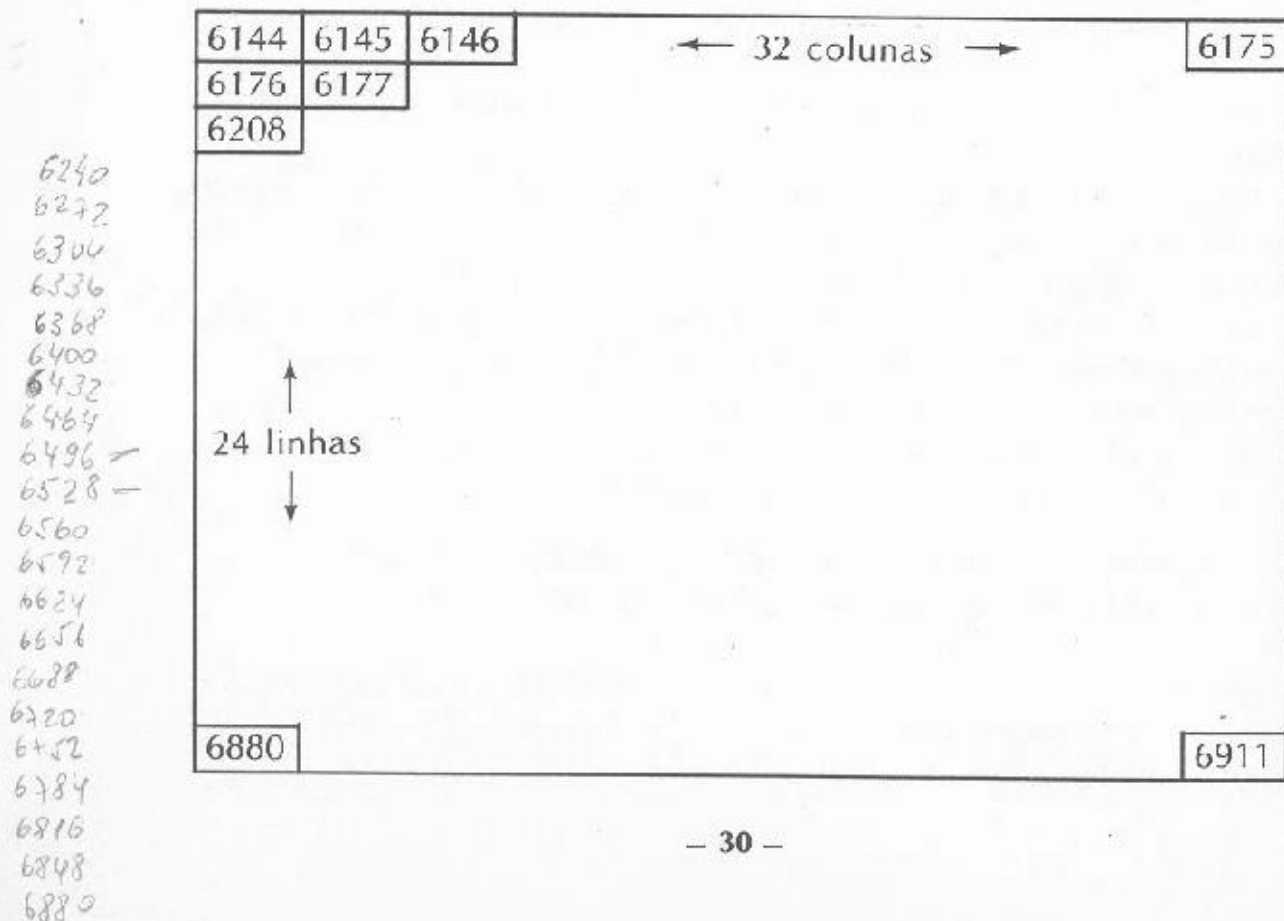
Agora vamos ver o que se passa quando se modificam os conteúdos destes bytes com o comando VPOKE:

```
10 FOR I = 2568 TO 2575 : READ J : VPOKE I , J : NEXT
20 DATA 48 , 48 , 48 , 48 , 48 , 48 , 72 , 132
```

Os bytes 2568 a 2575 são os que determinam a configuração da letra A. Nosso programa BASIC, modificando-os, vai portanto substituir esta letra pelo seguinte desenho:

De agora em diante, cada vez que teclarmos A, um pequeno foguete aparecerá na tela.

Naturalmente, o que acaba de ser realizado poderá efetuar-se em qualquer dos 256 caracteres estocados na tabela dos caracteres.

MODO TEXTO 32 COLUNAS (SCREEN 1)

Memória da tela (768 bytes)

Desta vez a tela permite a entrada de 32 caracteres na largura e 24 na altura. A memória da tela fica por conseguinte constituída de 768 bytes (32 * 24). Sua utilização é rigorosamente idêntica à que foi estudada no modo SCREEN 0. Observemos apenas que o primeiro byte tem por endereço 6144 e não 0.

VPOKE 6144, 67 : a letra C aparece em cima e à esquerda.
VPOKE 6175, 90 : a letra Z aparece em cima e à direita
VPOKE 6911, 50 : o algarismo 2 aparece em baixo e à direita

Evidentemente, a função VPEEK dá resultados previsíveis:

```
WIDTH ( 32 ) : CLS : LOCATE 2,1 : PRINT "MSX"
```

Pode-se verificar sem dificuldade que os bytes 6178, 6179 e 6180 contém os códigos ASCII das letras M, S e X.

Tabela dos caracteres (2 048 bytes)

Números dos bytes	Caracteres
0-1.....7	Caractere n° 0
8-9.....15	Caractere n° 1
16-17.....23	Caractere n° 2
...	
...	
...	
384-385.....391	Algarismo 0 : ASCII 48 (384=48*8)
392-393.....399	Algarismo 1 : ASCII 49 (392=49*8)
...	
...	
...	
520-521.....527	Letra A : ASCII 65 (520=65*8)
528-529.....535	Letra B : ASCII 66 (528=66*8)
...	
...	
...	
...	
776-777.....783	Letra a : ASCII 97 (776=97*8)
...	
...	
...	
2040-2041.....2047	Caractere n° 255

Ela é constituída, como na SCREEN 0, de 2 048 bytes mas seu endereço de base é igual a 0.

Digitemos de novo o programa do parágrafo precedente:

```
FOR I = 520 TO 527 : PRINT VPEEK ( I ) ; : NEXT
```

e executemo-lo. Obtemos o mesmo resultado:

```
32  80  136  248  136  136  0
```

Vamos poder redefinir aqui um caractere mas — esta é a diferença essencial com o modo 0 — nada nos impedirá de realizar este trabalho nos 8 bits de cada byte. Experimentemos:

```
FOR I = 520 TO 543 : VPOKE I , 255 : NEXT
```

Esta linha BASIC escreve o valor 255(11111111 binário) nos bytes 520 a 527. A letra A é então substituída por um quadradinho inteiramente colorido. A letra seguinte B (bytes 528 a 535) está sujeita à mesma regra, bem como a letra C (bytes 536 a 543). Quando você tentar digitar uma destas três teclas, verá aparecer na tela um caractere colorido.

Tabela das cores (32 bytes)

Se sob o controle de SCREEN 0, temos somente duas cores ao nosso dispor, uma para os caracteres e uma para o fundo, agora vamos ver que, dentro de certos limites, podemos escolher as nossas cores entre as 16 que o MSX possui.

Números dos bytes	Números dos caracteres em questão
8192	0 a 7
8193	8 a 15
8194	16 a 23
...	
...	
...	
8201	72 a 79
...	
...	
...	
...	
...	
8221	240 a 255

Como se vê, o byte 8192 se encarrega da cor da primeira classe de 8 caracteres. É preciso ficar bem claro: estes 8 caracteres podem ser coloridos com as cores que se quiser, mas não independentemente uns dos outros. Uma vez escolhida uma cor, ela será automaticamente atribuída ao conjunto dos caracteres cujos números se estendem de 0 a 7. Certo?

Isto valerá para todas as 32 classes de 8 caracteres.

Tomemos por exemplo o byte 8201: ele contém a cor das letras H (ASCII 72) a 0 (ASCII 79). Quando tivermos modificado o seu conteúdo, o fato de teclar uma das 8 letras em questão se fará com uma nova tonalidade, mas todos os outros caracteres serão exibidos na tela com as suas cores originais.

Pois muito bem, vejamos como agir para escolher as nuânces; você já sabe que as cores são numeradas de 0 a 15.

Cores	Código decimal	Código binário	Cores	Código decimal	Código binário
Transparente	0	0000	Vermelho	8	1000
Preto	1	0001	Vermelho-claro	9	1001
Verde	2	0010	Amarelo	10	1010
Verde-claro	3	0011	Amarelo-claro	11	1011
Azul-escuro	4	0100	Verde-escuro	12	1100
Azul-claro	5	0101	Magenta	13	1101
Vermelho-escuro	6	0110	Cinza	14	1110
Ciano	7	0111	Branco	15	1111

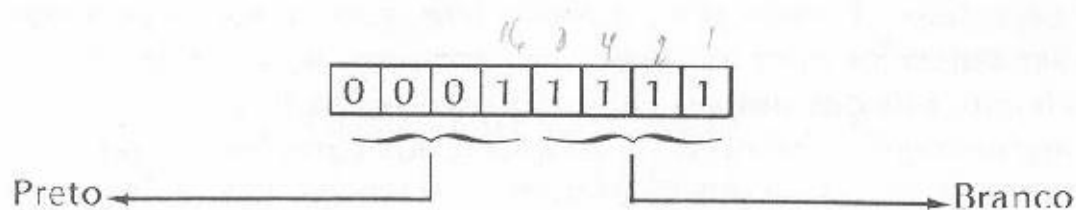
Se quisermos que um caractere seja exibido em vermelho sobre o amarelo, reunimos os quatro algarismos binários destas duas cores:

Vermelho: 1000 + Amarelo: 1010 = 10001010 binário, isto é, 138 decimal e escrevemos o valor 138 no byte que nos interessa:

```
VPOKE 8201 , 138
```

A partir daí, qualquer das letras, H, I, ..., 0 será exibida em vermelho sobre fundo amarelo.

```
VPOKE 8202 , 31
```

Os caracteres P, Q, ..., W aparecerão em preto sobre branco. Com efeito, 31 decimal é igual a:

MODO GRÁFICO (SCREEN 2)

Memória da tela (6 144 bytes)

← 32 colunas →

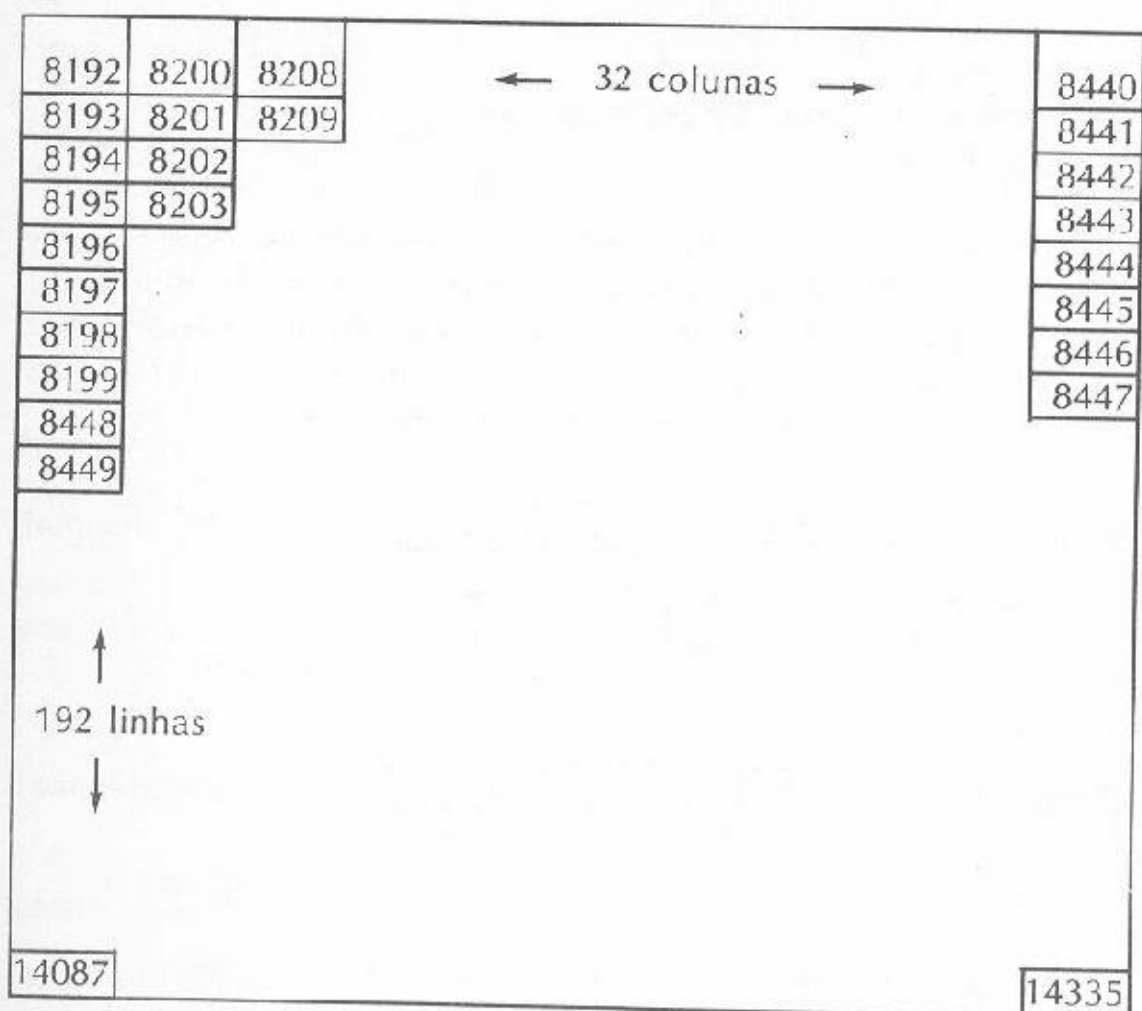
0, 8, 16 … 248
1, 9, 17 … 249
2, 10 … 250
3, 11 … 251
4 … 252
5 … 253
6 … 254
7 … 255
256
257

↑ 192 linhas ↓

5895 … 6143

Tabela das cores (6 144 bytes)

← 32 colunas →

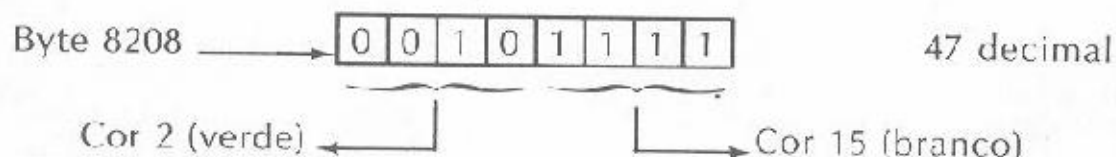
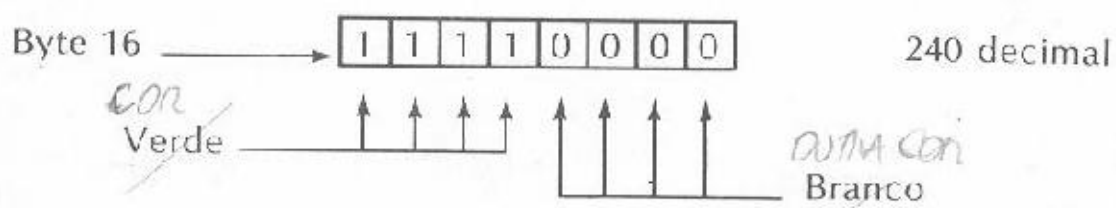
8192	8200	8208	...	8440
8193	8201	8209		8441
8194	8202			8442
8195	8203			8443
8196				8444
8197				8445
8198				8446
8199				8447
8448				
8449				
↑ 192 linhas ↓				
14087				14335

Neste modo de alta resolução, a tela sempre está dividida em 32 colunas mas cada coluna é constituída de 192 segmentos sobrepostos. Pode-se considerar que os segmentos 0 a 7 correspondem a um caractere do tipo SCREEN 1 (o caractere no alto da tela à esquerda), e que os segmentos 8 a 15 correspondem ao caractere do tipo SCREEN 1 colocado imediatamente à direita do precedente etc.

Multiplicando se por 8 o número de caracteres (24) que se pode fazer no sentido da altura na tela no modo SCREEN 1, obtêm-se 192 segmentos. O resultado está correto.

Observemos que a tabela de cores está em relação biunívoca com a memória da tela: a um byte da tela corresponde um e somente um byte de cor e o desvio que separa os endereços destes dois bytes é sempre igual a 8 192.

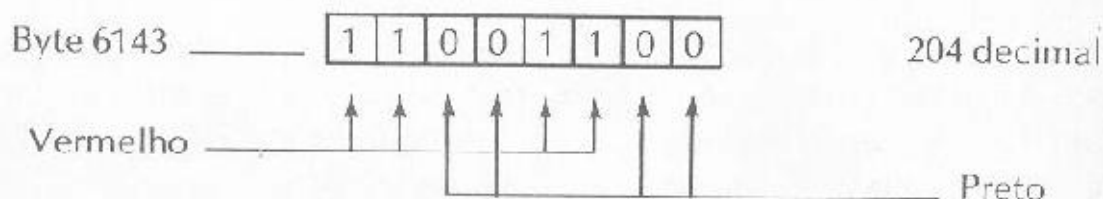
Os vários exemplos que seguem mostram que se pode agir sobre qualquer segmento da imagem:

```
10 SCREEN 2
20 VPOKE 16 , 240 : VPOKE 8208 , 47
30 GOTO 30
```

O byte nº 16 toma o valor 240; como em binário este número se escreve 11110000, deduz-se que os quatro pontos à esquerda do terceiro segmento da primeira linha da tela vão receber uma cor e os quatro pontos da direita uma outra cor. Quais são elas? São as inscritas no byte correspondente da tabela de cores.

Os quatro algarismos binários da esquerda do byte 8208, uma vez traduzidos em decimal, dão a cor dos pontos 1 no byte nº 16, por isso o nosso segmento aparece com a sua parte esquerda colorida de verde. Do mesmo modo, os quatro algarismos binários da direita do byte 8208 fornecem a cor (neste caso, branca) dos pontos 0 no byte de endereço 16.

A conclusão do que acaba de ser dito é a seguinte: o terceiro segmento da linha do alto da tela deve se tornar aparente com a sua metade esquerda colorida de verde e a metade direita de branco.

Um segundo exemplo; substituir a linha 20 por:

```
20 VPOKE 6143 , 204 : VPOKE 14335 , 129
```

Você já está entendendo bem? Os quatro algarismos do byte 14335 dão, depois de traduzidos do binário a cor 8 (isto é, vermelho). Isto quer dizer que cada vez que o computador encontrar o algarismo 1 no byte 6143, ele o colorirá de vermelho, e quando encontrar o algarismo 0, o colorirá de preto.

Byte 6143	vermelho	vermelho	preto	preto	vermelho	vermelho	preto	preto

Uma vez que resolvemos nos interessar pelo segmento colocado completamente embaixo e à direita do televisor, você deve ter diante dos olhos a confirmação do que acaba de ser explicado.

Um último exemplo:

```
20 FOR I = 248 TO 255 : VPOKE I , 16
25 VPOKE I + 8192 , 182 : NEXT
```

Os 8 bytes cujos endereços se seguem de 248 a 255 têm os seus conteúdos carregados a 16, isto é a 00010000 binário.

Os 8 bytes correspondentes (8440 a 8447) tomam o valor 182.

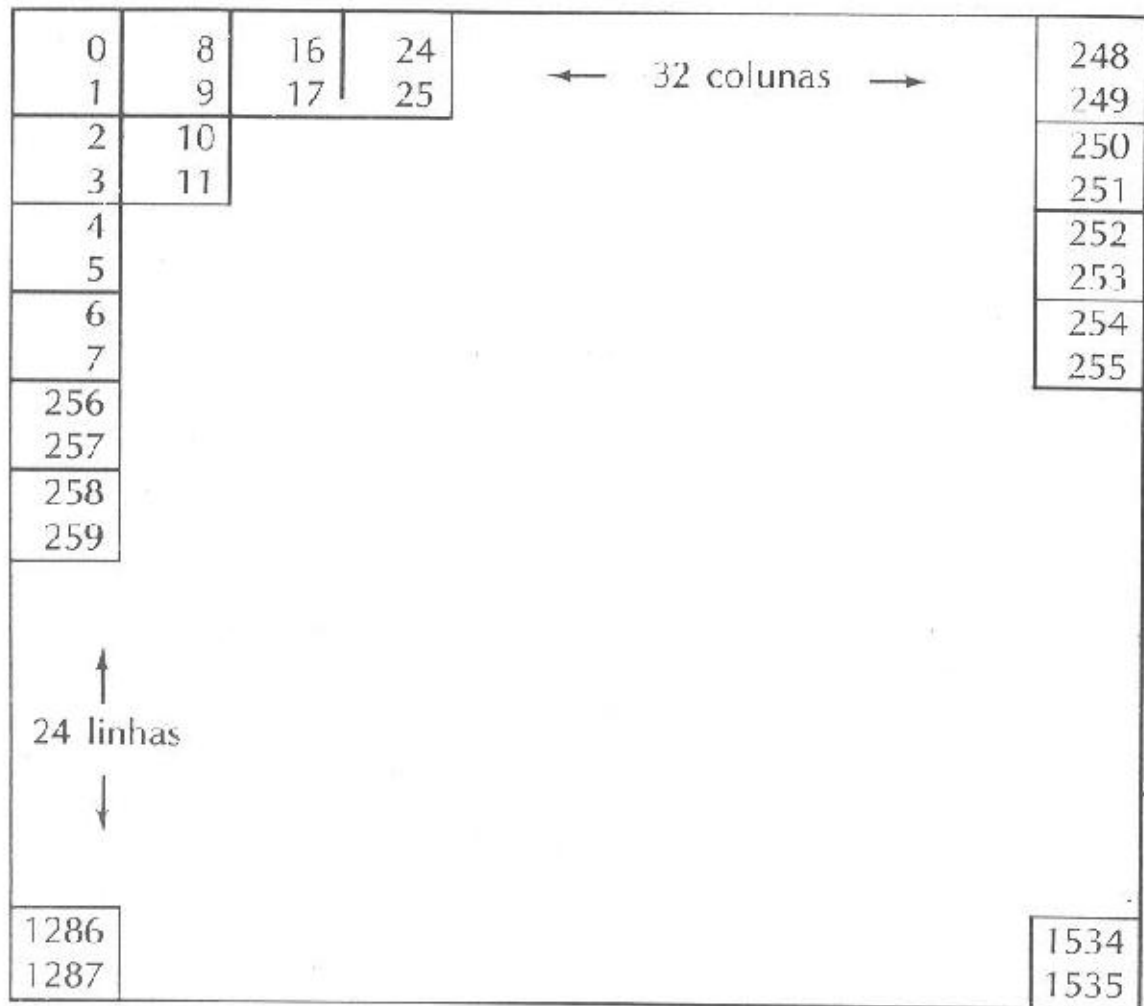
Assim temos cores dos 8 segmentos superpostos situados no alto e à direita da tela:

vermelho	vermelho	vermelho	amarelo	vermelho	vermelho	vermelho	vermelho

Memória da tela (1 536 bytes)

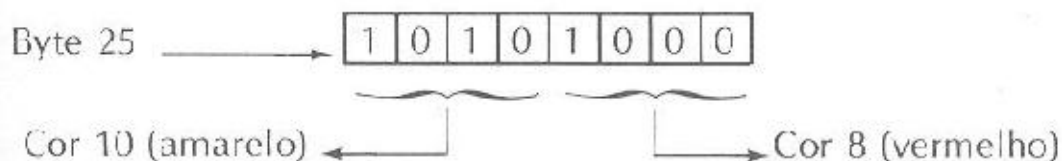
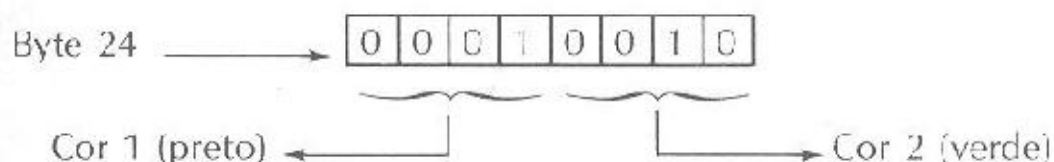
0 1	8 9	16 17	24 25	← 32 colunas →	248 249
2 3	10 11				250 251
4 5					252 253
6 7					254 255
256 257					
258 259					
↑ 24 linhas ↓					
1286 1287					1534 1535

Encontraremos os nossos caracteres do modo SCREEN 1, mas não se tratará mais de fazer aparecer caracteres: só poderemos colori-los com uma escolha de quatro cores para cada um.

Tomemos já um exemplo.

```
10 SCREEN 3
20 VPOKE 24 , 18 : VPOKE 25 , 168
30 GOTO 30
```

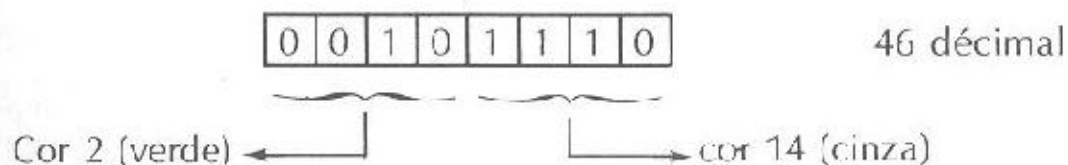
Você verá aparecer no alto do televisor o seguinte quadrado:

Preto 0001	Verde 0010	← Byte 24
Amarelo 1010	Vermelho 1000	← Byte 25

O byte 24 se ocupa da metade superior e o 25 da parte inferior. Agora, o sistema de codificação das cores já lhe é familiar:

Não há, neste modo, zona-de-memória reservada especialmente para as cores uma vez que o fato de se escrever um valor em um dos bytes 0 - 1535 se traduz diretamente pela coloração do meio-quadrado que lhe é associado. Como se vê, só temos duas cores disponíveis para byte: os quatro algarismos da esquerda determinam a cor do quadradinho da esquerda e os outros a cor do quadradinho da direita.

Segundo exemplo:

```
20 FOR I = 0 TO 248 STEP 8
25 VPOKE I , 46 : NEXT
```

Os bytes 0, 8, 16, 24, ... 248 recebem as cores correspondentes à decomposição do número 46 em duas partes binárias:

As partes superiores de todos os caracteres da primeira linha da tela tomam então as cores verde sobre cinza e vemos se suceder no

alto do televisor uma série de quadradinhos coloridos alternadamente de verde depois cinza.

PROGRAMA DE DEMONSTRAÇÃO

Para encerrar este capítulo, propõe-se uma comparação entre as velocidades de execução do mesmo programa, escrito em BASIC e em assembler. Apostemos que a comparação não dará vantagem à versão BASIC.

O objetivo das linhas que se seguem é iluminar cada um dos 6 144 segmentos da tela em modo 2.

```
10 SCREEN 2 : J = 0
20 FOR I = 0 TO 6143
30 VPOKE I , 255 : VPOKE I + 8192 , J
40 J = J + 16 : IF J = 256 THEN J = 0
50 NEXT : SCREEN 1 : LOCATE 0 , 10
60 PRINT " EIS AGORA A VERSÃO ASSEMBLER "
70 FOR I = 1 TO 2000 : NEXT
80 CLEAR 200 , 62000 : FOR I = 62001 TO 62030 : READ A$: POKE I ,
   VAL ( " &H " + A$ ) : NEXT : DEFUSR0 = 62001
90 SCREEN 2 : Y = USR0(X)
100 DATA 21 , 00 , 00 , 11 , 00 , 20 , 0E , 00 , E5 , 3E , FF , CD , CD , O7
110 DATA 19 , 79 , CD , CD , 07 , C6 , 10 , 4F , E1 , 23
120 DATA 7C , FE , 18 , 20 , EB , C9
130 GOTO 130
```

A parte que corresponde à versão BASIC compreende-se facilmente; ela é a aplicação do que foi estudado num dos parágrafos precedentes. Escreveu-se em cada um dos segmentos da memória da tela em modo 2 o valor 255. Este número é igual a 11111111 binário e em conseqüência todos os pontos dos diferentes segmentos serão iluminados. A cor deles dependerá do número contido pelo byte associado na tabela de cores.

Segmento 0 : 11111111	Byte 8192 (J = 0) : 00000000	8 pontos transparentes
Segmento 1 : 11111111	Byte 8193 (J = 16) : 00010000	8 pontos pretos
Segmento 2 : 11111111	Byte 8194 (J = 32) : 00100000	8 pontos verdes
Segmento 3 : 11111111	Byte 8195 (J = 48) : 00110000	8 pontos verde-claro

etc.

Observemos que as cores de fundo dos segmentos não têm a menor importância, pois neste programa todos os algarismos binários dos bytes da memória da tela valem 1.

As explicações para compreender a parte de linguagem de máquina deste programa são dadas agora. É inútil você se interessar por ela porque arriscaria a se desanimar diante do que pode parecer uma dificuldade intransponível. Passe portanto ao capítulo seguinte e esteja certo de que, em alguns dias, o que segue lhe parecerá muito claro.

Número de bytes: 30

Linhas	Códigos de Máquina	Assembler		
1	21 00 00		LD	HL,0
2	11 00 20		LD	DE,8192
3	0E 00		LD	C,0
4	E5	ROTINA	PUSH	HL
5	3E FF		LD	A,255
6	CD CD 07		CALL	1997
7	19		ADD	HL,DE
8	79		LD	A,C
9	CD CD 07		CALL	1997
10	C6 10		ADD	A,16
11	4F		LD	C,A
12	E1		POP	HL
13	23		INC	HL
14	7C		LD	A,H
15	FE 18		CP	24
16	20 EB		JR	NZ,ROTINA(−21)
17	C9		RET	

Linha 1: O registrador HL é carregado com o endereço do primeiro byte da memória da tela. Ele aparecerá sucessivamente sobre cada um dos bytes desta memória e na mesma ocasião sobre os bytes correspondentes da tabela de cores.

Linha 2: DE conservará o mesmo valor, 8192, durante todo o programa. Só servirá para ser acrescentado a HL para atingir a memória da cor.

Linha 3: O registrador simples C vai conter os valores 0, 16, 32, 48 etc. e indicará ao computador que número ele deve escrever nos bytes da cor.

Linhas 4 a 6: HL, que contém 0 por enquanto, é protegido na pilha, o acumulador é carregado com 255 e o subprograma de escrita na memória do vídeo é chamado. Acabamos de realizar em assembler o equivalente do comando BASIC: VPOKE 0,255.

Linha 7: Acrescenta-se 8192 a HL que aponta imediatamente sobre o byte da cor associado ao segmento 0.

Linha 8: O acumulador toma o valor do registrador C; logo, é ajustado para 0.

Linha 9: Eis uma nova chamada ao sistema para que ele escreva o algarismo 0 no byte apontado por HL. A linha BASIC correspondente será: VPOKE 8192, 0. O primeiro segmento da imagem tem, portanto, os seus 8 pontos iluminados com a cor... transparente.

Linhas 10 e 11: Acrescenta-se 16 ao conteúdo do acumulador e coloca-se de lado este valor (em C).

Linhas 12 e 13: HL encontra o seu valor de origem e se incrementa. Agora vale 1.

Linhas 14 a 16: Estabelecida a comparação entre o registrador H e o valor 24 que incide em números diferentes, o programa é deslocado de novo à linha ROTINA, 21 bytes para trás.

Por ocasião da segunda passagem na rotina, as operações seguintes serão executadas:

- escrita do valor 255 no byte nº 1;
- escrita do valor 16 no byte nº 8193.

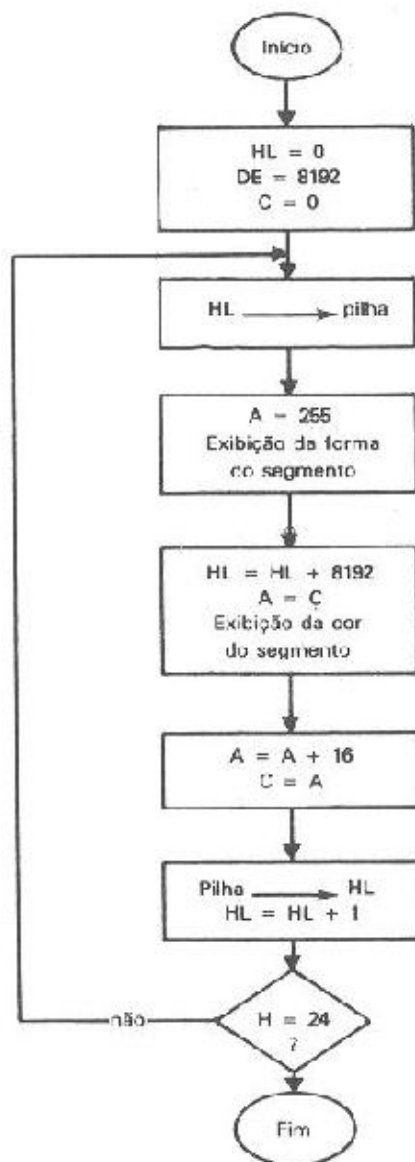
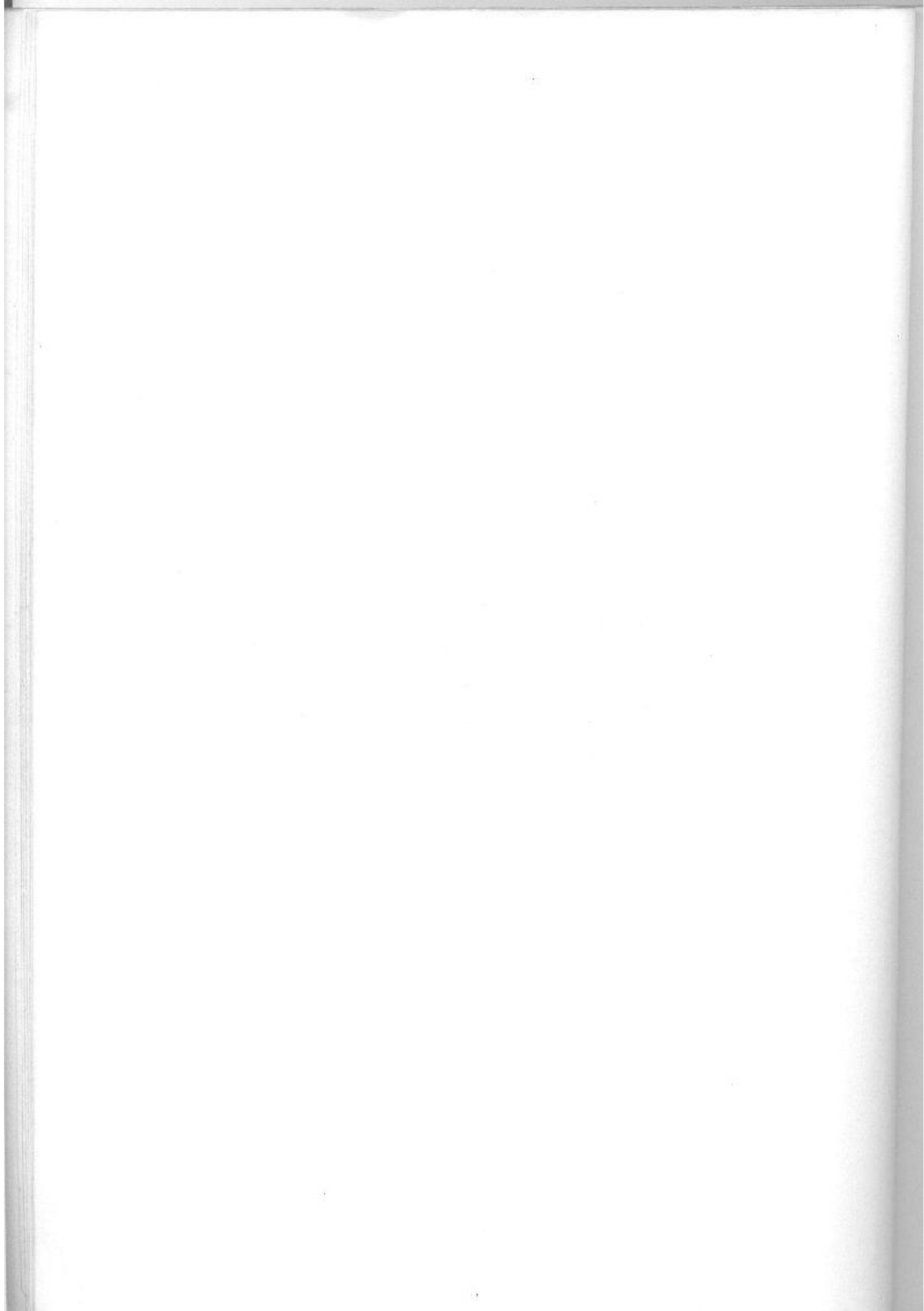
Um segmento colocado exatamente embaixo do precedente poderá então ser visível na tela. Será colorido de preto, uma vez que 16 se escreve 00010000 em binário.

Na terceira passagem pela linha ROTINA, um segmento de cor verde aparecerá embaixo dos dois primeiros.

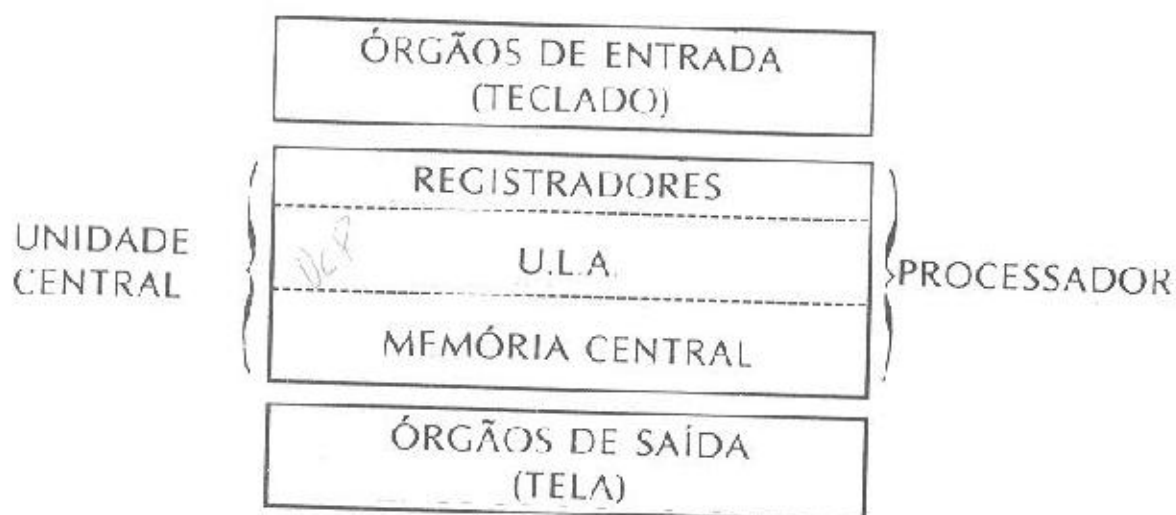
O programa fará o ciclo da rotina enquanto o registrador H não atingir o valor 24. Apliquemos a regra peso forte/peso fraco para ver qual será o valor de HL nesse momento:

256 * peso forte (registrador H) + peso fraco (registrador L) =
256 * 24 + 0 = 6144

Segue o fluxograma relativo a este programa:

Início
HL = 0
DE = 8192
C = 0
HL pilha
A = 255
Exibição da forma
do segmento
HL = HL + 8192
A = C
Exibição da cor
do segmento
A = A + 16
C = A
Pilha HL
HL = HL + 1
não
H = 24
?
Fim

3

A ARQUITETURA INTERNA DO MICROPROCESSADOR Z80

Antes de começar o estudo do microprocessador, é necessário situá-lo no seu meio. Assim este capítulo começará com algumas revisões sobre a estrutura geral de um microcomputador.

É essencialmente a memória central que será útil ao nosso estudo e é por esta razão que as memórias auxiliares não aparecem no esquema.

A MEMÓRIA CENTRAL

A memória central permite registrar, conservar e restituir a pedido as informações que lhe foram comunicadas. Estas informações são de dois tipos: dados ou programas. A priori, nada distingue em memória estes dois tipos de informações e somente a lógica do programa impedirá a confusão.

Por razões tecnológicas, a informação guardada na memória se encontra sob a forma de combinações dos algarismos binários 0 e 1. O bit (ou algarismo binário) é a unidade elementar de informação e só pode tomar um destes valores. Um byte é constituído de oito bits. O maior número que se pode ter num byte é o número 11111111 (ou seja 255 em decimal). O menor número é obtido quando os oito bits valem 0: é o número 00000000 (ou o 0 em decimal).

A memória central da maioria dos microcomputadores divide-se em 65 536 bytes e o computador é capaz de encontrar o conteúdo

de qualquer byte graças ao seu endereço. Os bytes da memória são numerados de 0 a 65 535 e é este número que se chama endereço. É possível saber o número contido por um byte com a função BASIC PEEK. Vamos experimentar:

PRINT PEEK (15000) ; resposta : 208

Uma vez que 208 se escreve 11010000 em binário, pode-se achar a configuração exata do byte nº 15000:

1	1	0	1	0	0	0	0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Pode-se dizer também que este byte contém o número hexadecimal D0.

Já que somos capazes de conhecer o valor que se encontra num byte, devemos ser capazes de modificar este valor: a instrução POKE está à nossa disposição para isto.

PRINT PEEK (40000) ; resposta : 255
POKE 40000, 100
PRINT PEEK (40000) ; resposta : 100

Antes da nossa intervenção, o byte nº 40000 continha o número 255. Todos os bytes contém um valor e é difícil dizer, em absoluto, a que ele corresponde. Inscrevemos por POKE o valor 100 no byte e só tivemos que pedir a confirmação com PRINT PEEK (40000).

Façamos uma nova experiência tomando o cuidado de não procurar inscrever em um byte um número superior a 255; isto teria como conseqüência a exibição na tela da mensagem *Illegal function call*. A razão disto é que com oito bits não podemos constituir um número superior a 11111111, ou seja, 255 em decimal.

Voltemos à nossa experiência:

PRINT PEEK (5000) ; resposta : 32
POKE 5000 , 100
PRINT PEEK (5000) ; resposta : 32

Pois bem, o comando POKE não nos permite modificar o conteúdo de qualquer byte da memória! Vamos saber a razão assim que tivermos distinguido os dois grandes tipos de memória.

A ROM (*Read Only Memory*) é a parte da memória central que podemos somente ler. Não é possível modificar um byte que se encontra naquela zona. O byte 5000 por exemplo se encontra na ROM

e por isso POKE não pode agir sobre ele. Quanto ao que concerne ao computador que é estudado neste livro, o MSX, é preciso saber que não poderemos de maneira alguma modificar um dos 32 767 primeiros bytes. Com efeito, esta é a parte da memória que contém o interpretador, isto é, o programa que vai traduzir o que se digita em BASIC para uma linguagem compreensível pela máquina. O conteúdo da ROM tem a vantagem de não se apagar quando se corta a corrente: é uma memória permamente.

A RAM (*Random Access Memory*), ou memória viva, contém todos os bytes que se podem ler — com PEEK — e também modificar — com POKE. O byte 40000, cujo conteúdo podemos modificar, encontra-se na zona viva da memória. O programa BASIC que digitamos é estocado na RAM, e sabemos que, assim que se desliga a força, o computador "esquece tudo". Isso põe em evidência uma propriedade da memória viva: ela é volátil, o seu conteúdo se perde assim que a unidade central não é mais alimentada.

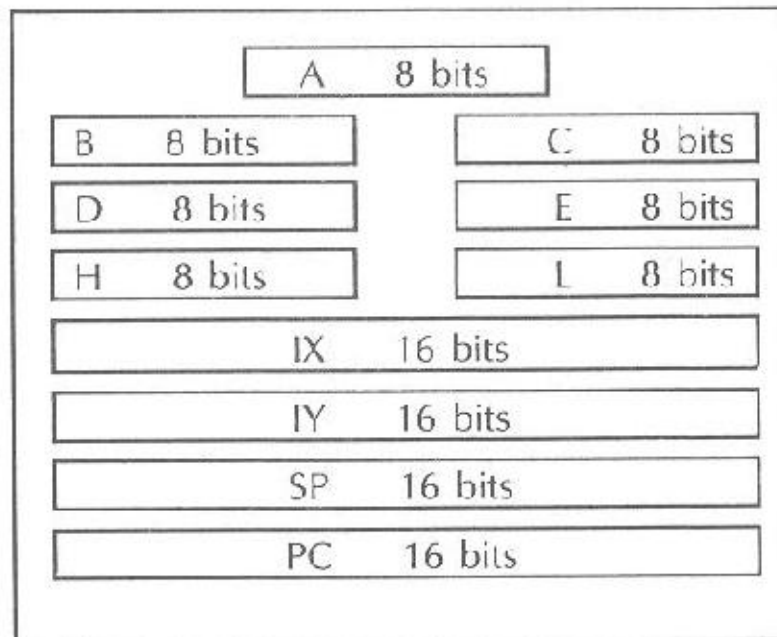
Vamos utilizar durante todo este livro as palavras PEEK e POKE. Estas instruções causam, às vezes, algumas surpresas; elas são consideradas indispensáveis por todos os espíritos curiosos que querem ir um pouco mais longe do que o BASIC. Que os leitores que ainda não se aventuraram se exercitem um pouco tentando escrever por POKE valores diferentes nos bytes 62382 a 62385: poderão facilmente ver os efeitos na tela.

A UNIDADE LÓGICA E ARITMÉTICA (ULA)

A ULA é constituída de circuitos eletrônicos e capaz de efetuar cálculos aritméticos (adições e subtrações) e escolhas lógicas (comparação de dois números). Ela possibilita também fazer operações de deslocamento e de rotação, operações às quais algumas páginas deste livro serão dedicadas mais adiante.

Escrevendo nossos programas, não teremos que intervir diretamente na ULA, mas, mesmo não sendo explicitamente o caso, não se deverá esquecer o papel capital desta unidade no computador: a ULA é a calculadora do microprocessador.

OS REGISTRADORES

	A 8 bits	
B 8 bits		C 8 bits
D 8 bits		E 8 bits
H 8 bits		L 8 bits
	IX 16 bits	
	IY 16 bits	
	SP 16 bits	
	PC 16 bits	

Os registradores são identificáveis a caracteres da memória pelos quais a informação vai transitar. O microprocessador Z80 contém registradores de 8 bits e registradores de 16 bits. As instruções BASIC não permitem o acesso a estes registradores e nunca poderemos com a função PEEK, por exemplo, saber o que contém este ou aquele registrador.

Os registradores A, B, C, D, E, H e L

Todos eles são constituídos de 8 bits, conseqüentemente o maior número que podem conter é 255 decimal (FF hexa ou 11111111 binário).

a. O registrador A

É o mais utilizado em assembler e freqüentemente dá-se-lhe o nome de acumulador. Seremos obrigados a passar por ele assim que, por exemplo, façamos um cálculo ou efetuemos uma comparação entre dois valores.

b. Os outros registradores de 8 bits

Não haveria nada de especial a dizer sobre estes registradores a não ser a particularidade muito importante de poderem ser associados dois a dois para constituir pares de registradores.

Vejamos isto de mais perto. Suponhamos que os bits dos registradores B e C estejam dispostos como segue:

	128	64	32	16	8	4	2	1
B	0	1	1	0	0	1	0	1

Em B se encontra portanto o número decimal 101 ou hexadecimal 65.

	128	64	32	16	8	4	2	1
C	1	0	0	0	1	1	1	1

Em C, é o número 143 (decimal) ou 8F (hexa) que se encontra. Para formar o registrador BC, associamos B e C:

B								C							
0	1	1	0	0	1	0	1	1	0	0	0	1	1	1	1

O valor de BC é então 0110010110001111, isto é, 25999 em decimal. Será necessário lembrar sempre que BC é um registrador de 16 bits e que corresponde portanto a dois bytes (ou seja, um número compreendido entre 0 e 65535).

O exemplo que acaba de ser dado seria também possível se substituíssemos os registradores B e C por D e E ou H e L. Teríamos então obtido os pares de registradores DE ou HL.

Os registradores IX e IY

São os registradores 16 bits e portanto pode-se escrever neles qualquer número de 0 a 1111111111111111 (16 vezes o algarismo 1). Se nos dermos o trabalho de traduzir este valor binário em decimal, obteremos 65535. Isto nos permite compreender o papel que IX e IY desempenharão: eles conterão geralmente um endereço de memória e este endereço poderá ser o de qualquer byte no intervalo 0-65535.

IX e IY são chamados freqüentemente de registradores de ende-

reços, em conseqüência do que acabamos de dizer; são chamados também de índice pois podemos utilizá-los no modo de endereçamento indexado (estudaremos isso em breve).

O registrador SP

Ele se chama ponteiro de pilha. Uma pilha é um lugar da memória onde serão estocados — empilhados — números uns em seqüência aos outros. A estrutura da pilha de um computador corresponde exatamente à de uma pilha de pratos: pode-se acrescentar um prato na pilha, mas, se quisermos retirar um, será sempre o último colocado que teremos que recuperar (último acrescentado, primeiro retirado).

Como a utilização da pilha não é evidente para o programador que dá os seus primeiros passos em assembler, vamos ver com calma um exemplo. Suponhamos que, no processador, os registradores BC e SP tenham os seguintes valores:

Os registradores B e C contém respectivamente os números decimais 224 (ou E0 hexa) e 241 (ou F1 hexa). Por este fato, o registrador BC contém o valor 57585 (E0F1 hexadecimal).

Em SP se encontra o número decimal 60000 (EA60 hexadecimal).

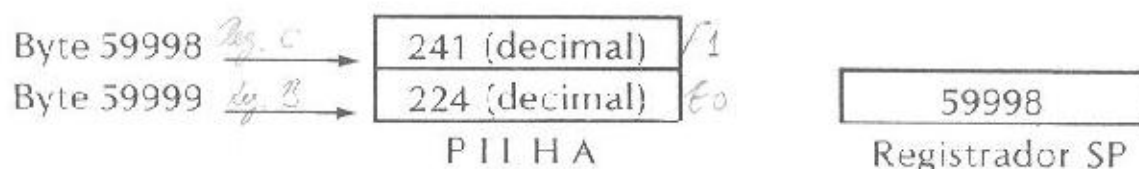
O valor 60000 indica, neste exemplo, que vamos empilhar os nossos números numa série de bytes da memória a partir justamente do byte 60000. Este mesmo byte está fora do nosso trabalho pois justamente no byte situado ao lado é que vai se dar a arrancada do nosso empilhamento.

PUSH BC

Encontramo-nos agora diante da nossa primeira instrução assembler. Ela utiliza a palavra inglesa PUSH (empurrar). As letras B e C esclarecem que é o conteúdo do registrador BC que vai ser colocado na pilha. Ora, não percamos de vista que a memória é consti-

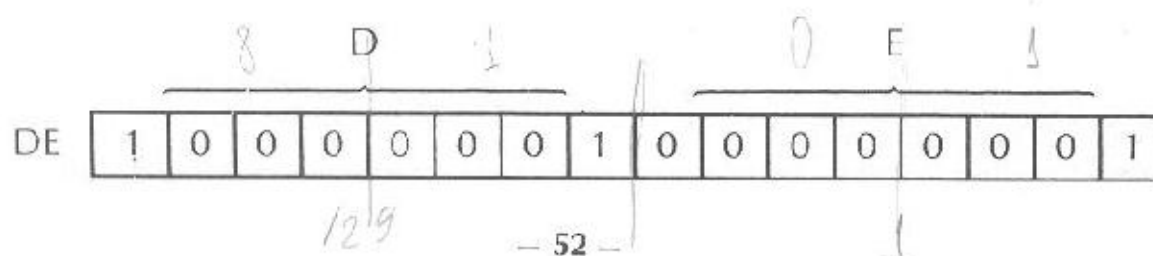
tuída por uma sucessão de bytes (8 bits) e que não se pode colocar o conteúdo do registrador BC (16 bits) num só byte. Independentemente disso, o valor do nosso registrador será guardado na memória em dois bytes consecutivos, como vamos ver.

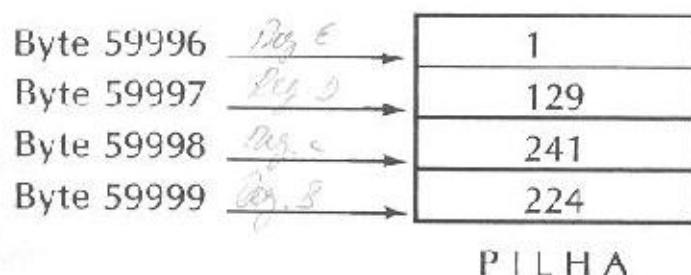
Quando a instrução PUSH BC tiver sido executada pelo microprocessador, eis o que terá acontecido:

- O registrador BC não terá sofrido nenhuma modificação. Seu valor, o número 57585, terá ido se escrever na pilha, mas este registrador terá conservado a sua identificação. Isso ocorrerá sempre que dermos ordem ao processador para transferir um número de um registrador para a memória: o registrador não será modificado e tudo se passará como se o registrador tivesse enviado somente um duplo, uma duplicata para a memória.
- A pilha será formada de dois bytes cujos conteúdos serão diretamente deduzidos do valor de BC. Os oitos bits da esquerda deste registrador (chamados bits de peso forte) serão recopiados no byte 59999 e os outros oito bits (ditos bits de peso fraco) serão inscritos no byte 59998. Assim tudo se passará como se tivéssemos empilhado primeiramente o registrador B e em seguida o registrador C.

Como se vê, o registrador SP agora vale 59998. Está portanto definido o seu papel: ele contém o endereço do último byte no qual um valor foi empilhado. Diz-se que ele aponta para o ponto mais alto da pilha.

Continuemos com:

```
PUSH    DE
```

Desta vez é o conteúdo do registrador DE que vai se colocar na pilha — isto é, nos bytes 59997 e 59996. Imaginemos, para fixar as idéias, que DE tenha a seguinte configuração:

O registrador de peso forte D vale então 129 (decimal) e o de peso fraco vale 1. O conteúdo de DE é por conseguinte igual a 33025.

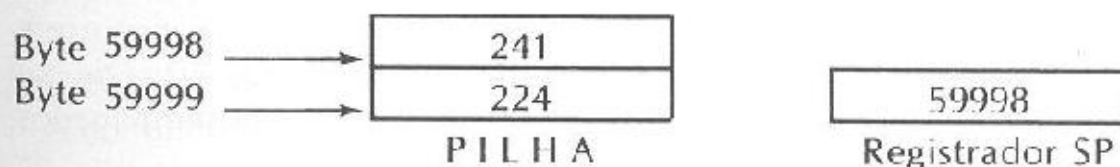
Após a execução da instrução de empilhamento pelo computador, a pilha fica assim constituída:

PILHA

O registrador SP vai portanto apontar para o byte 59996.

SP | 59996 |

Antes de passar à operação de desempilhamento, observemos que os números são estocados na pilha sobre bytes cujos endereços vão decrescendo cada vez mais. Não se pode alterar isto; sem dúvida é a própria lógica do nosso computador que empilha para baixo. O importante é saber isto.

```
POP     DE
```

É a nossa segunda instrução assembler: ela efetua exatamente o trabalho inverso da primeira. Ela vai procurar um número na pilha e o escreve no registrador DE. Logo, ela desempilhou os bytes 59996 e 59997 e recolocou-os em DE. Por isso mesmo a nossa pilha só contém dois bytes e o registrador SP repassa a 59998.

O interesse das instruções PUSH e POP não é imediatamente claro para os programadores que estão começando a estudar assembler, pois eles não vêem para que servem duas coisas que não fazem nada além de nos levar de volta ao nosso ponto de partida. E, no entanto, todo o interesse reside aí: a principal dificuldade do assembler decorre do fato de que dispomos apenas de um número muito reduzido de registradores. Veremos em breve a enorme vantagem de colocar o valor de um registrador na pilha (PUSH), utili-

zar este registrador para fazer outra coisa e encontrar (POP) o valor inicial deste mesmo registrador.

Terminemos o nosso exemplo:

POP   DE

A máquina vai escrever nos registradores E e D os números 241 e 224. Ela executa muito corretamente a ordem que lhe é dada tomando os números escritos no alto da pilha — portanto 241 e 224 — e colocando-os em E e D. Ela não se preocupa em saber de que registrador provêm estes dois valores. Neste momento, os dois registradores BC e DE contêm precisamente o mesmo valor: 57585. A pilha que tínhamos constituído está reduzida a zero e o registrador SP retoma o seu valor inicial 60000.

No final das contas, no nosso exercício, o registrador BC não teve em nenhum momento o seu valor modificado, mas uma cópia do seu conteúdo terá sido transferida para o registrador DE.

Outro exemplo agora:

Suponhamos que os pares de registradores BC, DE e HL contenham os valores decimais 1000, 2000 e 3000.

Quais serão os valores de cada um deles quando a série de instruções tiver sido executada?

PUSH BC — PUSH DE — PUSH HL
POP   BC — POP   HL — POP   DE

Resposta : BC = 3000, HL = 2000; DE = 1000

Será preciso, nos programas assembler, ter sempre presente que a pilha é utilizada de modo interno pelo microprocessador. Teremos problemas se, por descuido, não recolocarmos a pilha no estado em que a encontramos. Se tivermos necessidade, por exemplo, de empilhar seis bytes na pilha, deveremos estar certos de que no fim do programa os seis bytes em questão foram desempilhados.

O registrador PC

É um registrador de 16 bits que se chama contador de programa ou contador ordinal. O número que está escrito neste registrador é o endereço da próxima instrução a executar. Ele permite ao microprocessador saber sempre por qual byte do programa deverá se

interessar. Só o utilizamos em programação nos casos muito particulares: ele não é diretamente acessivel.

OS MODOS DE ENDEREÇAMENTO

Um modo de endereçamento é um meio que permite ao microprocessador ter acesso a um dado. Este dado pode ser um número qualquer do qual teremos necessidade no programa, um número que já se encontra em um registrador, ou ainda um número que se encontra escrito em alguma lugar da memória.

O conhecimento dos principais modos de endereçamento é obrigatório: ele permite escrever os programas do modo mais curto, mais simples e mais legível possível.

O endereçamento inerente

O endereçamento inerente é habitualmente reservado para as instruções que agem diretamente sobre os valores contidos pelos registradores. Estas instruções se compreendem automaticamente e não há nenhuma necessidade de acrescentar indicações.

Exemplo: CPL

Todos os microprocessadores compreendem este gênero de instrução: ela significa que o registrador A (e só ele e não outro) será complementado. De modo bem claro, isto quer dizer que cada um dos seus bits tomará outro valor binário; os algarismos 1 serão substituídos por algarismos 0 e reciprocamente.

A continha 143 decimal (por exemplo), ou seja 10001111 binário.

CPL

A contém 112 decimal, isto é, 01110000 binário.

O endereçamento imediato

Neste modo, um valor aparece depois da instrução assembler. Tomemos por exemplo: LD A, 100.

A fórmula LD, que se encontrará no decorrer de todo este livro, significa que se vai colocar (carregar) um número no registrador. É fácil ver que aqui a instrução LD não pôde ser escrita automaticamente como no endereçamento inerente. Temos que acrescentar obrigatóriamente indicações em seguida, e se quisermos colocar um número no acumulador A, é preciso dizer qual é ele.

No modo de endereçamento imediato, será escrito o valor marcado após a vírgula (aqui 100), no registrador que se tiver escolhido escrevendo o seu nome antes da vírgula. No nosso exemplo, é o registrador A que está em questão, mas qualquer outro registrador 8 bits ou 16 bits também serviria.

A continha, por exemplo, 50:

```
LD   A,100
```

Logo A contém 100.

Um contra-exemplo:

```
LD   A,300
```

Em princípio, esta linha deveria escrever em A o número 300. Você já tinha adivinhado, isto não tem lógica uma vez que A é um registrador 8 bits e o número máximo que pode conter é 255.

O endereçamento extenso

É freqüentemente chamado de endereçamento absoluto por ser um modo que permite-se ter acesso diretamente ao conteúdo de qualquer byte da memória.

```
LD   A,(10)
```

O acumulador será carregado não com o número 10, como ocorreria no endereçamento imediato, mas com o valor escrito no byte nº 10.

```
PRINT PEEK(10) ; resposta : 38
```

No final das contas, o registrador A conterá 38. Os parênteses que encerram o valor 10 são usados para evitar a confusão entre o modo imediato e o extenso. Obedeceremos a este tipo da grafia em todo o livro e ela terá sempre o mesmo significado: quando um valor estiver entre parênteses, ele corresponderá ao endereço de

um byte da memória e é o conteúdo deste byte que deverá ser considerado.

Segundo exemplo:

```
LD   A,(5000)
```

Uma cilada, como no parágrafo precedente? 5000 parece de importância mínima para o nosso acumulador 8 bits. Mas ao contrário, esta linha não é um contra-senso: ela significa que o registrador A vai conter não o número 5000 mas o número que está escrito no byte cujo endereço é 5000.

```
PRINT PEEK(5000) ; resposta : 32
```

o que faz com que A seja carregado com o valor 32, assim que seja executado a instrução assembler.

O endereçamento registrador

Este modo de endereçamento não faz aparecer nenhum número na instrução e só é utilizável quando se quer intervir nos conteúdos dos registradores. Exemplo:

```
INC   B
```

Esta instrução dá ao computador a ordem de incrementar o registrador B, isto é, aumentar o seu valor em uma unidade.

B continha 200 (por exemplo).

```
INC B
```

B contém 201.

Um outro exemplo:

```
LD   C,D
```

Encontramos o mnemônico de carregamento LD. Não há nenhuma dificuldade para interpretar esta instrução. Se 20 e 30 são, por exemplo, escritos nos registradores C e D, a execução da instrução inscreverá o valor 30 em C (e D ficará inalterado).

O endereçamento indireto

É conhecido como o endereçamento extenso, mas a expressão

que se encontra entre parênteses é, desta vez, o nome de um registrador.

Vejamos alguns exemplos.

LD C,(HL)

O registrador simples C vai ser carregado com o valor que se encontra no byte que tem por endereço o número escrito em HL.

Suponhamos que, neste registrador 16 bits, haja o número 20 000.

PRINT PEEK (20000) ; resposta : 199

Após LD C,(HL), haverá no registrador C o valor 199. Esta instrução é por conseguinte completamente equivalente a LD C,(20000) (endereçamento extenso).

Segundo exemplo:

LD HL,25000
LD B,(HL)

É preciso considerar que a primeira instrução LD coloca diretamente no registrador HL o número 25 000 (modo de endereçamento imediato). A segunda instrução vai recopiar no registrador B o conteúdo do byte 25000.

PRINT PEEK (25000) ; resposta . 8

Logo, B conterá o valor 8.

Façamos uma nova experiência:

LD C,(IX+10)

Desta vez, o processador vai procurar, para escrever em C, o valor que se encontra em memória no byte que tem por endereço o número contido em IX ao qual acrescentamos 10. É mais fácil compreender do que explicar!

Admitamos que IX contenha o valor 26000. Acrescentamos 10 a 26000 e o referido byte é então o nº 26010.

PRINT PEEK (26010) ; resposta : 40

Assim, neste último exemplo, C vai ser carregado com o número 40.

Um último exemplo:

LD H,(IY−20)

Considerando que IY contém o número 27 000 (por exemplo), es-

ta instrução indica à máquina que ela deve colocar em H o número escrito no byte 26080.

Se sabermos que o PEEK deste byte vale 200, deduzimos que o registrador H será carregado com o valor 200.

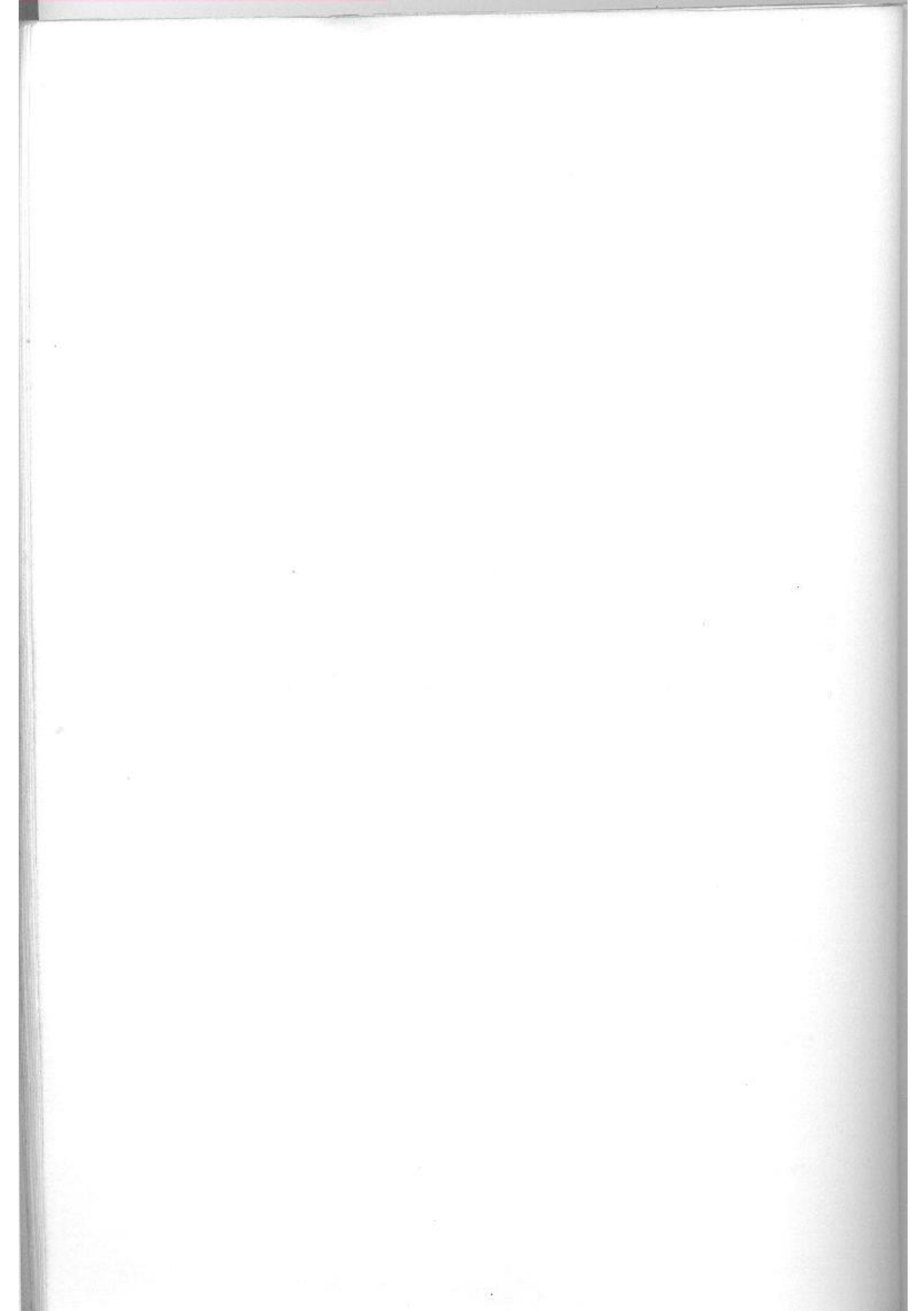
Observemos que só os registradores IX e IY têm o privilégio de poder aceder a um caractere de memória cujo endereço é dado por seus conteúdos aos quais se acresce ou se retira uma determinada quantia. As instruções do tipo:

```
LD   A,(BC+5)
```

ou

```
LD   A,(DE-3)
```

não existem. É por esse motivo que às vezes consideramos que o endereçamento indireto, quando relativo aos registradores IX e IY, deve ter um nome diferente. Fala-se então de endereçamento indexado.

Terminamos este capítulo e o assunto difícil da teoria já acabou. Vamos nos sentar diante do nosso MSX e vejamos como obrigá-lo a compreender e em seguida executar um programa escrito em assembler.

4

ESTUDO DE UM EXEMPLO

Digite e execute no seu MSX o seguinte programa:

```
10 CLEAR 200 , 62000 : FOR I = 62001 TO 62007 : READ A $ :
   POKE I , VAL ( " &H " + AS ) : NEXT : DEFUSR0 = 62001
20 DATA 21 , DB , F3 , 3E , 00 , 77 , C9
30 Y = USR0 ( X )
```

Em seguida, tente apertar uma das teclas. O ruído que habitualmente acompanha tal ação desapareceu; digitar as teclas não tem mais nenhum eco. Este estado de coisas não foi provocado, como de costume, pela modificação de um dos parâmetros da instrução SCREEN ou por um POKE. Nós o conseguimos graças à execução do nosso primeiro programa escrito em linguagem de máquina.

POKE 62427,0

Linhas	Códigos de máquina	Assembler		
1	21 DB F3	LD	HL,62427	(imediato)
2	3E 00	LD	A,0	(imediato)
3	77	LD	(HL),A	(indireto)
4	C9	RET		(inerente)

Esta apresentação será retomada ao longo deste livro. Por enquanto, façamos uma análise minuciosa de tudo o que é novo.

PARTE ASSEMBLER

É a que o leitor assimilará mais depressa.

LD HL,62427

Encontramos uma instrução já vista. Ela significa que o registrador HL vai conter o valor decimal 62427. Vamos decompor este número para obter o peso forte e o fraco. O detalhe deste trabalho não será mais retomado no resto do livro.

62427/256 = 243 (divisão inteira)

cujo peso forte é igual a 243 (ou seja F3 hexadecimal).

62427 − (256 ∗ 243) = 62427 − 62208 − 219

O peso fraco é portanto igual a 219 (ou seja, DB hexadecimal). Obtemos a igualdade:

62427 = 243 * 256 + 219

isto é:

Peso forte * 256 + peso fraco.

Isto nos dá a configuração do registrador HL:

Peso forte 243								Peso fraco 219							
1	1	1	1	0	0	1	1	1	1	0	1	1	0	1	1
Registrador H (F3 hexa)								Registrador L (DB hexa)							

do que se deduz que os registradores H e L contêm respectivamente os valores 243 e 219. Teremos necessidade destas indicações um pouco mais à frente. Quanto à nossa instrução, guardemos simplesmente que o modo de endereçamento utilizado é o modo imediato.

```
LD  A,0
```

Desta vez, trata-se do registrador A. Ignora-se que número se encontrava no acumulador (não esqueça que é o nome do registrador A) antes desta instrução, mas agora estamos certos de que o seu conteúdo é igual a 0.

```
LD  (HL),A
```

O número colocado em A (portanto 0) é recopiado não no registrador HL mas no byte apontado por HL. Visto que este registrador contém o valor 62427, o conteúdo do byte nº 62427 é levado a 0. As três primeiras instruções assembler são por conseguinte equivalentes à linha BASIC:

```
POKE 62427 , 0
```

Desta forma se explica porque o nosso computador ficou sem voz. Cada vez que se anula o conteúdo do byte 62427, o mesmo fenômeno se produz, o bip desaparece. Você poderá recuperar o eco tradicional escrevendo por POKE qualquer valor diferente de 0 no byte 62427 ou tentando modificar o programa assembler para que ele mesmo dê novamente voz ao MSX. E, enquanto isto, penso no meio de escrever o programa não em quatro linhas mas em três.

RET

Esta instrução, freqüentemente colocada na última linha, serve para indicar ao microprocessador que o programa assembler terminou. Então o BASIC retoma o controle da situação.

CÓDIGOS DE MÁQUINA

Um computador só compreende números e para ele as expressões LD ou RET não significam nada. Logo, vai ser preciso traduzir o programa que escrevemos em assembler para a única forma compreensível para o computador: o código de máquina.

Insistimos bastante na diferença que há entre a linguagem assembler e a de máquina: a primeira é concebida para a mente humana e é formada de instruções que têm sentido para nós. Quando escrevemos LD A,0, sabemos muito bem o que se passará no processador; não temos necessidade de fazer um grande esforço para compreender que o conteúdo do registrador A será modificado e substituído por um novo valor. A forma assembler permite escrever programas legíveis, programas escritos com palavras ou observações cujo sentido aprenderemos depressa. Quem de nós será capaz de interpretar a série dos códigos de màquina 21, DB, F3, 3E... que, além do mais, são escritos em hexadecimal?

Vamos, portanto, à tradução para linguagem de máquina do programa. É um exercício que se pode dizer fácil, mas cuidado com o menor erro de codificação que seria em seguida muito difícil de encontrar!
Utilize a tabela do Apêndice D.

21 é o equivalente de LD para a máquina. Você constatará que LD se codifica de diferentes maneiras segundo o modo de endereçamento e os referidos registradores. O registrador que nos interessa é HL e o modo de endereçamento é o imediato. Será necessário lembrar sempre que os códigos são escritos em hexadecimal, 21, bem como todos os outros códigos desta tabela, respeita esta regra.

DB e **F3** são, como já vimos, o peso fraco e o forte do número 62427 O microprocessador, após ter interpretado 21, esperará que lhe digamos com que número ele deve carregar HL. Uma vez que DB e F3 vêm em seguida a 21, ele compreenderá que o valor F3DB (ou

62427 decimal) deve ser colocado no par de registradores. Vamos observar, por um momento, uma particularidade do Z80. Este processador tem o hábito de inverter a ordem lógica do peso fraco e do forte. Para ele, 62427 se escreve DB seguido de F3 e não o contrário. Não tente fazê-lo mudar de idéia, ele terá sempre a última palavra. Então, vale mais tomar nota das suas manias e não se esquecer delas.

3E é o código de máquina da instrução LD A no modo imediato. Quando o computador lê este código, ele sabe que deve interessar-se pelo valor seguinte. Com efeito, se o conteúdo do acumulador deve ser modificado, é necessário saber com que número.

00 é o número em questão. O registrador A contém agora o valor nulo.

77 depois de interpretado, leva ao desaparecimento do *bip* sonoro. O microprocessador compreendeu, efetivamente, que devia escrever o valor 0 (contido por A) no byte apontado por HL, isto é, no byte 62427.

C9 é a tradução em linguagem de máquina da instrução RET. Este número termina a série dos códigos de máquina e passa a vez ao BASIC.

PROGRAMA BASIC

CLEAR é uma instrução que permite reservar lugar na memória. Ela é utilizada para se fornecer ao programa um lugar suficiente nas manipulações de cadeias (200 é o número habitual de bytes) e também para indicar ao computador que ele não deverá se servir de um determinado número de bytes da memória.

```
CLEAR 200 , 62000
```

deve ser interpretado do seguinte modo: a máquina poderá utilizar qualquer byte cujo endereço for inferior ou igual a 62000, mas não deverá de forma alguma modificar os conteúdos dos bytes 62001, 62002, 62003 ... 62335. Este último número é o endereço do fim da memória útil do MSX.

```
FOR I = 62001 TO 62007
```

Precisamos destes sete bytes e estamos certos, graças a CLEAR, de que eles estarão reservados para o nosso uso.

READ A$

A leitura dos sete valores escritos em DATA será efetuada; é necessário fazer ler uma cadeia de caracteres, pois, de um lado, letras aparecem na linha 20 e de outro, os próprios algarismos são escritos em hexadecimal e não em decimal.

POKE I , VAL (" &H " + A$)

No início da rotina, I vale 62001 e A$ vale 21. O interpretador reconhece portanto o número hexadecimal &H21 e escreve o seu valor no byte 62001. Na segunda passagem, o número DB será colocado no byte 62002. Após a última passagem, os sete códigos de máquina serão inscritos nos bytes que vão de 62001 a 62007. Compreende-se assim porque o número F3DB (62427 em decimal) foi escrito em duas partes colocadas, uma no byte 62002 (parte fraca) e a outra no byte 62003 (parte forte).

NEXT

O programa é carregado na memória central e só falta executá-lo.

DEFUSR0 = 62001

Esta instrução indica ao BASIC em que endereço (aqui 62001) deve começar a execução do programa em linguagem de máquina. É possível fazer coabitar na memória do computador até 10 subprogramas em código de máquina. Seus pontos de entrada serão definidos por DEFUSR0, DEFUSR1, ... DEFUSR9.

Y = USR0 (X)

De maneira idêntica a RUN que lança um programa BASIC, Y = USR0(X) desencadeia a execução do programa de máquina. O comando das operações passa ao microprocessador que vai realizar as instruções correspondentes aos valores que se encontram nos bytes 62001 e seguintes. Encontrando 21, ele vai carregar o registrador HL com o número que se encontra nos dois bytes seguintes e continuar assim até o valor C9. Considere, num primeiro tempo, que as variáveis Y e X não servem para nada, são ficticias. Teremos oportunidade mais adiante de lhes encontrar uma utilidade.

OBSERVAÇÃO

Pode-se perguntar qual é o papel dos bytes cujos endereços vão de 62008 a 62335. Ele é nulo; estes bytes não são utilizados nem pelo computador, já que CLEAR não o impede, nem por nós, pois, em conclusão, o nosso programa só tem necessidade de sete bytes. Pela lógica teríamos que escrever o BASIC com as seguintes modificações:

```
10 CLEAR 200 , 62328 : FOR I = 62329 TO 62335 ... :
   DEFUSR0 = 62329
```

A linha 10 teria colocado nos bytes 62329 a 62335 os sete códigos de máquina que o processador teria encontrado com DEFUSR0. Escolhemos bloquear todos os bytes a partir de 62001 por uma razão muito simples: quase todos os programas deste livro estarão alojados na memória a partir deste endereço a fim de ter uma apresentação uniforme. E este primeiro programa obedece esta regra. Acrescentemos que a perda de 300 bytes não deve ser considerada como excessiva e que apesar de tudo é mais fácil, se temos um programa de 15 bytes, por exemplo, escrever:

```
FOR I = 62001 TO 62015
```

do que

```
FOR I = 62321 TO 62335
```

UTILIZAÇÃO DO EDITOR/ASSEMBLER ZEN

Se você comprou a fita que permite programar diretamente o MSX em assembler, este parágrafo guiará os seus primeiros passos.

Coloque a fita no leitor associado ao seu computador e digite a seguinte linha BASIC:

```
CLEAR 200 , 40959 : BLOAD "ZEN"         (ENTER)
```

O gravador começa a funcionar e o programa escrito na fita é transferido para a memória central. A tarefa deste programa será traduzir (em nosso lugar) os comandos assembler em códigos. Po-

deremos, sob o controle deste programa, digitar diretamente uma instrução como por exemplo LD HL,62447. O computador nos fornecerá automaticamente os três códigos correspondentes: 21, DB e F3 e, além disso, ele os escreverá nos bytes da memória que tivermos escolhido.

Comecemos pelo começo.

A instrução CLEAR 200, 40959 agora nos é familiar. Ela vai impedir que o BASIC utilize os bytes cujo endereço é superior a 40959. Isto é necessário pois o programa que provém da fita está justamente carregado na memória a partir do byte nº 40960. Desde que este carregamento é efetuado, abandona-se o BASIC

```
DEFUSR1 = 40960 : Y = USR1(X)
```

A palavra ZEN seguida do sinal > é então visível na tela e o computador espera as nossas ordens.

Teclemos ENTER para que a tela se apague e digitemos sucessivamente as teclas E e ENTER. Imediatamente um número de linha aparece. Só falta executar um programa teclando ENTER no fim de cada linha.

```
1   ORG    62001
2   LOAD   62001
3   LD     HL,62427
4   LD     A,0
5   LD     (HL),A
6   RET
7   END
8   .
```

Você reconheceu o programa estudado desde o início deste capítulo. Todavia quatro linhas foram acrescentadas, são as seguintes:

```
1   ORG    62001
2   LOAD   62001
7   END
8   .
```

As duas primeiras indicam ao computador que ele deve gerar os códigos e escrevê-los nos bytes sucessivos da memória. O primeiro deles terá por endereço 62001.

Os dois últimos servem para delimitar o fim do programa. Quando o ponto (seguido de ENTER) da linha 8 tiver sido teclado, você reencontrará a mensagem ZEN>

Aperte então todas as teclas A (depois ENTER) e V (depois ENTER). Durante um breve instante, o computador carrega o nosso programa: as nossas instruções assembler são então traduzidas para linguagem de máquina. Assim que este trabalho termina, vê-se aparecer na tela a listagem do programa seguinte:

```
                      ORG    62001
                      LOAD   62001
F231     21DBF3       LD     HL;62427
F234     3E00         LD     A,0
F236     77           LD     (HL),A
F237     C9           RET
                      END
```

A parte da direita é a que nós demos entrada, A parte central corresponde aos códigos 21, DB, F3 etc. A coluna da esquerda não é outra coisa senão a tradução hexadecimal dos números compreendidos entre 62001 e 62007.

Resumindo: o program ZEN traduziu as instruções assembler em uma série de códigos e inscreveu-se estes códigos nos 7 bytes 62001, 62002, ... 62007. OK? Falta-nos compreender como utilizar estes dados. Digite B (para Bye) depois ENTER. Você está transferindo neste momento o controle para o BASIC.

```
10 DEFUSR0 = 62001 : Y = USR0(X)
```

Faça executar este programa curto. O bip familiar que acompanha a batida das teclas terá desaparecido. A instrução USR0 mandou o microprocessador executar os códigos escritos a partir do endereço 62001. Estes códigos, como sabemos, foram colocados aí pelo programa ZEN e não sofreram modificação quando o BASIC foi novamente chamado.

E para terminar: é perfeitamente possível encontrar o nosso programa assembler, pois a volta ao BASIC não apaga a memória do MSX. O caminho a seguir é o seguinte:

```
Y = USR1(X)              (ENTER)
A mensagem  ZEN >     aparece
Digite Pn                (ENTER)
```

P é a abreviatura de PRINT e n é o número de linhas que se deseja fazer escrever. No nosso exemplo, será necessário digitar P7, considerendo-se que o nosso programa comporta 7 linhas. A totalidade da listagem será exibida na tela.

5

ELEMENTOS DE PROGRAMAÇÃO DO Z80

Nota importante

O leitor não deverá se esquecer, em cada um dos programas que são dados a título de exemplo, de incluir as linhas 10 e 20 seguintes:

```
10 CLEAR 200 , 62000 : FOR I = 62001 TO 620?? : READ A$ :
   POKE I , VAL ( "&H" + A$ ) : NEXT : DEFUSR0 = 62001
20 DATA ...
```

Os dois pontos de interrogação da linha 10 deverão ser substituídos pelo número de bytes do programa em linguagem de máquina e a linha 20 deverá conter em DATA a lista dos códigos de máquina escritos sob a forma hexadecimal.

LD A

Abreviação de LoaD A (carregar A), esta instrução permite colocar um valor de 8 bits no registrador A.

Exemplo: MODO DE ENDEREÇAMENTO IMEDIATO (6 bytes)

Programa BASIC

```
10 CLEAR 200 , 62000 : FOR I = 62001 TO 62006 : READ A$ :
   POKE I , VAL ( " &H " + A$ ) : NEXT : DEFUSR0 = 62001
20 DATA 3E , 00 , 32, AB , FC , C9
30 Y = USR0 ( X )
```

As linhas 10 e 20 são dadas neste exemplo mas só os seus números aparecerão no resto do livro. Tome o cuidado a cada vez de completá-las.

Programa assembler

Linhas	Códigos de máquina	Assembler		
1	3E 00	LD	A,0	(imediato)
2	32 AB FC	LD	(64683),A	(extenso)
3	C9	RET		(inerente)

A apresentação do programa em forma de tabela só é feita pela intenção de clareza. Ao contrário do BASIC, os números de linhas não têm nenhuma importância para o microprocessador que, neste exemplo, se interessa somente pela seqüência dos bytes 62001 a 62006.

Linha 1: O valor 0 (00 hexadecimal ou 00000000 binário) é colocado no acumulador A. 3E é o código hexadecimal da instrução LD A no modo imediato.

Linha 2: O conteúdo do registrador A é guardado no byte 64683. 32 é o código de máquina da instrução, AB (171 decimal) e FC (252 decimal) são o peso fraco e o forte do número 64683.

64683 = 252 × 256 + 171

Lembre-se de que o microprocessador se ocupa sempre primeiro do peso fraco.

Linha 3: O computador é avisado do fim do programa assembler e a volta ao BASIC é programada.

Você não vê nada de especial na tela? Então experimente apertar uma tecla e verá que agora tudo aparece na tela em letras minúsculas, embora a luz vermelha tenha ficado acesa no console.

A explicação deste fato é simples: o byte 64683 contém um valor que indica à máquina o tipo de impressão. Se este valor for igual a 255, a impressão na tela é realizada em maiúscula, e se ela for nula, os caracteres aparecem em minúscula. Ora, qual é o objetivo do nosso programa assembler se não o de inscrever o valor 0 no byte 64683?

Para obter o funcionamento normal da tecla Caps Lock, basta simplesmente você apertá-la duas ou três vezes. Tudo voltará à ordem.

LD A, (BC)

Esta instrução escreve no acumulador o valor contido pelo byte cujo endereço está inscrito no registrador BC.

Exemplo: MODO DE ENDEREÇAMENTO INDIRETO (8 bytes)

Programa BASIC

```
10
20
30 POKE 62100 , 1 : Y = USR0 ( X )
40 REM
50 J = J + 1 : IF J < 10 THEN 40
60 POKE 62100 , 0 : Y = USR0 ( X )
```

Programa assembler

Linhas	Códigos de máquina	Assembler	
1	01 94 F2	LD	BC,62100
2	0A	LD	A,(BC)
3	32 C4 F7	LD	(63428),A
4	C9	RET	

Primeira chamada do programa assembler: linha BASIC 30.

Linha 1: O registrador duplo BC é carregado no modo imediato com o número 62100. Naturalmente os registradores HL ou DE teriam igualmente servido para isso.

Linha 2: O conteúdo do byte indicado por BC é colocado no acumulador. Isto quer dizer que A, agora, se acha carregado com o conteúdo do byte 62100. Ora, a leitura da linha BASIC 30 nos indi-

ca que escrevemos por POKE o valor 1 neste byte. Logo, A vale 1. De acordo?

Linha 3: Uma cópia do conteúdo do registrador A é enviada ao byte 63428. Este byte tem a particularidade de estar a serviço do par de instruções TRON/TROFF. Se ele vale 1, o que é o caso, o comando TRON é ativado e por isso os números das linhas executadas pelo BASIC imediatamente começam a aparecer na tela. É claro que as linhas 40 e 50 não têm nada a ver com o nosso programa e só estão aqui para nos mostrar claramente a ação de TRON.

Linha 4: O BASIC retoma a direção do programa e executa 10 vezes a rotina 40 — 50 deixando cada vez no televisor a indicação da linha que está executando.

Segunda chamada do programa assembler: linha BASIC 60.

O computador é lançado de novo no programa de máquina. Este realiza as mesmas funções como da primeira vez, com a diferença de que vai achar no byte 62100 o valor 0. Quando ele o tiver recopiado no byte 63428, o processo de escrita automática dos números de linhas será desativado. TRON será substituído por TROFF.

Acabamos de realizar uma passagem de informação entre o BASIC e o assembler. Esta operação foi efetuada graças ao auxílio do byte 62100, mas compreenda bem que poderíamos nos ter servido de qualquer um dos seus similares que pertencem ao intervalo 62009 — 62335.

LD (IX + n)

Esta instrução permite ler ou escrever um valor de 8 bits no byte cujo endereço é calculado acrescentando ao conteúdo do registrador IX o valor n.

Exemplo: MODO DE ENDEREÇAMENTO INDIRETO (19 bytes)

Programa BASIC

```
10
20
30 Y = USR0 ( X ) : PRINT PEEK ( 62102 )
```

Programa assembler

Linhas	Códigos de máquina	Assembler	
1	FD 21 94 F2	LD	IY,62100
2	DD 21 D0 07	LD	IX,2000
3	DD 46 EC	LD	B,(IX – 20)
4	DD 4E 0A	LD	C,(IX + 10)
5	0A	LD	A,(BC)
6	FD 77 02	LD	(IY + 2),A
7	C9	RET	

Este é um programa que tem unicamente a utilidade de lhe exigir um pouco de ginástica mental e de lhe proporcionar o domínio dos modos de endereçamento do Z80. Portanto, antes mesmo de ler as explicações e fazer executar o programa, tente prever qual será a resposta dada pelo computador na linha 30. Considere que você só tem à disposição a função PEEK e faça com ela experiências de leitura da memória que você quiser.

Linhas 1 e 2: Os pares de registradores IY e IX contém no início do programa os valores 62100 e 2000. Seus conteúdos não serão modificados. Quanto ao assunto da codificação de máquina, observe que a instrução sozinha corresponde aos valores hexadecimais FD e 21. Freqüentemente teremos oportunidade de achar instruções assembler que deverão se codificar em dois bytes.

Linha 3: No registrador B está colocado o valor do byte 1980 (2000 − 20). Vejamos o que realmente contém este byte.

PRINT PEEK (1980) ; resposta : **33**

Devemos agora dedicar algumas linhas à maneira como a instrução é codificada.

DD 46 é a tradução (ainda em dois bytes) de LD B;
EC corresponde ao valor negativo −20 codificado no modo complemento da base dois que se obtém, lembre-se, invertendo todos os bits do número 20 e acrescentando uma unidade.

0 0 0 1 0 1 0 0 ←——— 20 decimal
1 1 1 0 1 0 1 1 ←——— complemento da base 2
1 1 1 0 1 1 0 0 ←——— acréscimo de 1

Ora, 11101100 é igual a 236 decimal ou EC hexadecimal.

Linha 4: O conteúdo do byte 2010 (2000 + 10) é transferido no registrador C. Conheceremos o valor de C digitando:

PRINT PEEK (2010) ; resposta : 227

Linha 5: Sabemos o que contém os registradores B e C; a regra peso forte/peso fraco vai nos fornecer o valor de BC

256 * 33 (registrador B) + 227 (registrador C) = 8675 (registrador BC).

O acumulador, portanto, é carregado com o conteúdo do byte apontado por BC, isto é, o byte 8675.

PRINT PEEK (8675) ; resposta : **60**

Linha 6: Só falta escrever este valor num lugar da memória (byte 62102) de modo que o BASIC o encontre.

LD — Resumo

Seguem-se as diferentes possibilidades oferecidas ao programador na utilização da instrução LD (LOAD).

LD	A,n8	LD	BC,n16
LD	B,n8	LD	DE,n16
LD	C,n8	LD	HL,n16
LD	D,n8	LD	SP,n16
LD	E,n8	LD	IX,n16
LD	H,n8	LD	IY,n16
LD	L,n8		
n8 é um número de 8 bits (registrador simples) que vale no máximo 255. n16 é um número de 16 bits (registrador duplo) que vale no máximo 65535.			

Exemplo:

LD	A,25	A contém o valor 25
LD	H,50	H contém o valor 50
LD	DE,40000	

DE contém o valor 40000. Esta instrução é totalmente equivalente às duas linhas seguintes:

LD	D,156	(peso forte de 40000)
LD	E,64	(peso fraco de 40000)

LD	R,R'	LD	SP,HL
R e R' são qualquer um dos registradores A, B, C, D, E, H ou L.		LD	SP,IX
		LD	SP,IY

Exemplo:

```
LD   B,100
LD   H,B
LD   L,B
LD   SP,HL
```

A execução deste programinha coloca em B, H e L o mesmo valor decimal 100. Deduz-se então o valor comum dos registradores HL e SP: 100 * 256 + 100 = 25700.

LD R,(HL)	LD (HL),R
R é qualquer um dos registradores de 8 bits A, B, C, D, E, H ou L.	

Exemplo:

```
LD   B,50
LD   HL,40000
LD   (HL),B
LD   C,(HL)
```

Este programa escreve em B o número 50, em HL o número 40000, no byte nº 40000 o número 50 e no registrador C o número 50. De acordo?

LD A, (endereço)
LD A,(BC)
LD A,(DE)
	LD (endereço),A
	LD (BC),A
	LD (DE),A

O acumulador é o único registrador de 8 bits que pode corresponder com um byte da memória por intermédio de um endereço (endereçamento extenso) ou dos registradores indicadores BC e DE (endereçamento indireto).	

Exemplo:

```
LD   BC,60000
LD   A,(BC)
INC  A
LD   (BC),A
```

Primeira e segunda linhas: BC é carregado com o número 60000 e A com o conteúdo do byte 60000.

Terceira e quarta linhas: Este conteúdo é incrementado e reescrito no byte 60000. Este último terá portanto o seu valor aumentado de uma unidade.

LD (n16),Rd	LD Rd,(n16)
n16 é um número de 16 bits e Rd é qualquer um dos pares de registradores entre os seguintes: BC, DE, HL, SP, IX ou IY.	

Exemplo:

```
LD  DE,30000
LD  (50000),DE
```

Inscrevendo 30000 em DE, atribui-se ao registrador D o valor 117 (peso forte) e ao registrador E o valor 48 (peso fraco). Naturalmente, o conteúdo do registrador DE não corre o risco de entrar no único byte nº 50000, o seguinte deverá portanto ser utilizado. Não se esqueça de que o microprocessador escreve primeiro o peso fraco — 48 no byte 50000 — e o peso forte em seguida — 117 no byte 50001.

LD (HL),n8
LD (IX+n),n8
LD (IY+n),n8

O número de 8 bits n8 é escrito no byte cujo endereço é contido por HL ou deduzido de IX ou IY.

Exemplo:

```
LD  IX,40000
LD  (IX+5),100
```

O valor 100 se encontra no byte de endereço 40005. Observemos que se n é negativo, deve-se anotá-lo sob a forma "complemento da base 2".

LD (IX+n),R LD (IY+n),R	LD R,(IX+n) LD R,(IY+n)
R é qualquer um dos registradores de 8 bits A, B, C, D, E, H ou L.	

Exemplo:

```
LD  C,150
LD  IY,60000
LD  (IY+10),C
LD  E,(IY+20)
```

Sendo C e IY respectivamente carregados com os números 150 e 60000, deduz-se que o byte 60010 tem seu conteúdo levado a 150. Quanto ao registrador E, ele se encontra com o valor do byte 60020 (aliás este programa não diz qual é este valor).

Antes de passar à instrução seguinte, traduza com calma para códigos de máquina os exemplos que acabamos de dar e execute os programas correspondentes. Não se esqueça de que a função BASIC PEEK dá o conteúdo de qualquer byte da memória. Sirva-se dele para verificar que o microprocessador escreveu certo o número certo, no byte certo.

CALL

Esta instrução chama um subprograma e indica ao microprocessador em que endereço o subprograma deve iniciar. O retorno se efetua na instrução que segue CALL.

Exemplo: MODO DE ENDEREÇAMENTO ABSOLUTO (11 bytes)

Programa BASIC

```
10
20
30 CLS : LOCATE 10 , 10 : Y = USR0 ( X )
```

Programa assembler

Linhas	Códigos de máquina	Assembler
1	3E 41	LD A,65
2	CD A2 00	CALL 162
3	3E 42	LD A,66
4	CD A2 00	CALL 162
5	C9	RET

Este é um programa chave para o nosso assunto. De um lado, vai nos permitir aprofundar os nossos conhecimentos sobre a forma como funciona o nosso computador; de outro, vai nos fazer compreender como podemos, em assembler, imprimir letras na tela.

Linha 1: O acumulador é carregado com um número que é o código ASCII da letra maiúscula A.

Linha 2: É a parte central do programa. Pode-se comparar CALL com a instrução BASIC GOSUB: o microprocessador vai começar

a executar os códigos que se encontram a partir do endereço 162 (00A2 hexa). Você têm dificuldade em compreender, você não vê nenhum significado nestes números, você acha que não está claro? Falemos francamente: no nível de adiantamento do nosso estudo, isto não pode estar claro. É mesmo um buraco negro. Ignora-se que tipo de instrução o computador vai executar! Então, o que se deve reter de tudo isso?

- Primeiramente, que o número 162 não foi escolhido ao acaso, ele é parte de uma zona de memória da máquina que é reservada ao sistema. Você vai se interessar por isso mais tarde; a documentação fornecida com o MSX lhe será útil nessa ocasião.
- Em segundo lugar, que a partir do momento em que um programa se desloca a este endereço, uma letra aparece na imagem. A sua colocação é dada pela instrução LOCATE da linha BASIC 30: é aproximadamente o centro da tela.
- Em terceiro lugar, que o caractere que é exibido relaciona-se diretamente com o conteúdo do registrador A. O subprograma de exibição sempre faz aparecer o caractere cujo código ASCII se encontra no acumulador. É por isso que a letra A está presente na tela.
- Em quarto lugar, que após ter realizado o trabalho, o processador encontra de novo a instrução que segue imediatamente CALL. No nosso caso, trata-se da linha 3.

Linhas 3 e 4: Renova-se a operação e vê-se aparecer desta vez a letra B ao lado da precedente. É fácil ver que, aí, o registrador A foi carregado com o código ASCII da segunda letra do alfabeto previamente ao chamado da rotina.

Terminaremos o nosso estudo com duas observações. De um lado, a instrução CALL se endereça diretamente a um byte da memória e não utiliza nenhum registrador. Por esta razão reencontramos um modo conhecido e não falamos de endereçamento absoluto. De outro lado, o subprograma de exibição desloca sempre automaticamente o cursor de um caractere para a direita. Isso explica porque a letra B é escrita ao lado da letra A.

ADD A

Esta instrução vai somar os conteúdos do registrador A e de um byte-memória. O resultado da adição será então colocado em A.

Exemplo: MODO DE ENDEREÇAMENTO INDIRETO (14 bytes)

Programa BASIC

```
10
20
30 INPUT " PRIMEIRO NÚMERO " ; N : POKE 62100 , N
40 INPUT " SEGUNDO NÚMERO " ; M : POKE 62101 , M
50 Y = USR0 ( X )
60 PRINT " A SOMA VALE " ; PEEK (62102) : GOTO 30
```

Programa assembler

Linhas	Códigos de máquina	Assembler	
1	DD 21 94 F2	LD	IX,62100
2	DD 7E 00	LD	A,(IX+0)
3	DD 86 01	ADD	A,(IX+1)
4	DD 77 02	LD	(IX+2),A
5	C9	RET	

Linha 2: O registrador A é carregado com o valor contido no byte apontado por IX, isto é, o byte 62100. A linha BASIC 30 escreveu neste byte a primeira parcela da soma. N não deve, é evidente, ultrapassar 255, senão a mensagem *Illegal function Call* aparece na sua tela.

Linha 3: ADD A,(IX + 1) significa que se soma o valor de A e o valor inscrito no byte que tem por endereço IX + 1. Por isso esta linha adiciona o primeiro e o segundo número da soma, tendo este último sido colocado por POKE no byte 62101 (linha BASIC 40).

Linha 4: Estando o resultado da operação no acumulador, só falta guardar a soma obtida no byte 62102 uma vez que é aí que o basic irá buscar a resposta.

Façamos rodar o programa.

Primeiro número	Segundo número	Soma
10	20	30
100	0	100
200	60	4
250	250	244

Nada a criticar nos dois primeiros casos: os resultados estão conformes com as nossas previsões. Quanto aos cálculos seguintes, não será muito complicado estabelecer a sua coerência: uma vez que A é um registrador de 8 bits, o maior número que se pode escrever nele é 11111111 (255 decimal). Se, nesta ocasião, tentarmos somar 1, o registrador passará a 00000000 (0 decimal) e é o que explica que 260 tornou-se 4 na nossa terceira soma. De maneira análoga, somando 250 e 250, não encontramos 500, mas 500 − 256, ou seja 244.

DIFERENTES FORMAS DA ADIÇÃO DE 8 BITS

ADD	A,n8	n8 é um número de 8 bits.
ADD	A,R	R é um dos registradores A, B, C, D, E, H ou L.
ADD ADD ADD	A,(HL) A,(IX+n) A,(IY+n)	

ADD 16 bits

ADD permite também somar os valores inscritos nos dois pares de registradores. Pode-se portanto adicionar dois valores de 16 bits. O único modo de endereçamento possível é o endereçamento registrador.

Exemplo: MODO DE ENDEREÇAMENTO REGISTRADOR (12 bytes)

Programa BASIC

```
10
20
30 INPUT " PRIMEIRO NÚMERO " ; N
40 N1 = INT ( N / 256 ) : POKE 62101 , N1
50 N2 = N - N1 * 256 : POKE 62100 , N2
60 INPUT " SEGUNDO NÚMERO " ; M
70 M1 = INT ( M / 256 ) : POKE 62103 , M1
80 M2 = M - M1 * 256 : POKE 62102 , M2
90 Y = USR0 ( X )
100 PRINT " A SOMA VALE " ; 256 * PEEK (62105) + PEEK
    (62104)
```

Programa assembler

Linhas	Códigos de máquina	Assembler	
1	ED 4B 94 F2	LD	BC,(62100)
2	2A 96 F2	LD	HL,(62102)
3	09	ADD	HL,BC
4	22 98 F2	LD	(62104),HL
5	C9	RET	

Linha 1: O registrador BC é carregado com o primeiro número N. Considerando-se o fato de que BC é um registrador de 16 bits, a instrução LD vai procurar na memória o valor dos 2 bytes 62101 (peso forte) e 62100 (peso fraco). Supondo-se N igual a 1000, observemos como isso se passa:

N1 = INT (1000/256) ou seja N1 = 3. O byte 62101 contém o número 3.

N2 = 1000 − 3 * 256 ou seja N2 = 232. O byte 62100 contém o número 232.

Após a execução da linha, BC se apresenta sob a forma seguinte:

0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0

byte de peso forte (3) — byte de peso fraco (232)

Aplicando, a título de verificação, a fórmula

256 * peso forte + peso fraco

obtém-se de fato

256 * 3 + 232, isto é, o número 1000.

Linhas 2 e 3: A operação é efetuada. O primeiro número N já se encontra em BC e o segundo, M, deve ser procurado na memória. Escolhamos, por êxemplo, M igual a 2000. As linhas BASIC 70 e 80 colocaram o valor de M nos bytes 62102 e 62103 da seguinte forma: a parte de peso forte de 2000, isto é, 7 (M1), é inscrita no endereço 62103 e a sua parte fraca, 208 (M2) no endereço 62102.

Podemos resumir as linhas 1, 2 e 3 dizendo que tudo se deu como se tivéssemos somado diretamente os números N e M escritos nos endereços 62100 e 62101 de um lado, 62102 e 62103 do outro lado. Não existe infelizmente instrução que o faça de maneira direta e a passagem pelos registradores BC e HL foi obrigatória.

Linha 4: Tendo sido colocado em HL o resultado da adição, só falta guardar o conteúdo deste registrador num local de memória onde a linha BASIC 100 poderá procurá-lo. Uma vez que este resultado é um número de 16 bits, a instrução LD vai colocá-lo em dois bytes, a saber, a parte fraca no endereço 62104 e a parte forte no endereço 62105.

Fazendo executar o programa em alguns exemplos, você obser-

vará que, cada vez que a soma ultrapassar 65535, o registrador HL a faz recomeçar de zero. 65535 é com efeito o maior valor decimal que se pode escrever em 16 bits.

PRIMEIRO NÚMERO ? 50000
SEGUNDO NÚMERO ? 20000
A SOMA VALE 4464, ou seja, 70000 − 65536.

DIFERENTES FORMAS DA ADIÇÃO DE 16 BITS

ADD HL,Rd	Rd é um dos pares de registradores BC, DE, HL ou SP.
ADD IX,Rd	Rd é um dos pares de registradores BC, DE, IX ou SP.
ADD IY,Rd	Rd é um dos pares de registradores BC, DE, IY ou SP.

SUB 8 bits

SUB é uma instrução que permite diminuir do conteúdo do acumulador A um valor de 8 bits. O resultado da operação é escrito em A. Os modos de endereçamento imediato, registrador e indireto são utilizáveis.

Exemplo: MODO DE ENDEREÇAMENTO INDIRETO (14 bytes)

Programa BASIC

```
10
20
30 INPUT " PRIMEIRO NÚMERO " ; N : POKE 62100 , N
40 INPUT " SEGUNDO NÚMERO " ; M : POKE 62101 , M
50 Y = USR0 ( X )
60 PRINT " A DIFERENÇA VALE " ; PEEK (62102)
```

Programa assembler

Linhas	Códigos de máquina	Assembler	
1	DD 21 94 F2	LD	IX,62100
2	DD 7E 00	LD	A,(IX+0)
3	DD 96 01	SUB	(IX+1)
4	DD 77 02	LD	(IX+2),A
5	C9	RET	

Linha 1: O registrador IX é carregado no modo imediato com o número 62100. É preciso considerar que o primeiro termo da diferença é justamente guardado nesse endereço e que o segundo é guardado no byte seguinte.

Linha 2: LD A,(IX + 0) signiica que se coloca no acumulador o número que se encontra no byte 62100 (IX + 0): é a linha BASIC 30 que anteriormente inscreveu neste byte o número N.

Linha 3: Diminuiu-se de N o número M contido no byte 62101 (IX + 1). O número M é conhecido do programa assim que a linha BASIC 40 é executada.

Observemos que a subtração se faz sempre no mesmo sentido: é o byte da memória que é diminuído do acumulador e não o contrário. Além disso, observaremos que a escrita da instrução não faz aparecer o nome do registrador A. Isso seria inútil, pois o único registrador pelo qual transitam todas as subtrações é precisamente A.

Linha 4: Já que o resultado da operação está agora em A, só falta estocar este resultado em um local da memória determinado que o programa BASIC poderá encontrar. A linha 60 manda procurar a resposta no byte 62102 (IX + 2).

DIFERENTES FORMAS DA SUBTRAÇÃO DE 8 BITS

SUB	R	Diferença efetuada entre o acumulador e qualquer um dos registradores A, B, C, D, E, H ou L.
SUB SUB SUB	(HL) (IX+n) (IY+n)	Diferença efetuada entre o acumulador e o byte da memória indicado.
SUB	n8	Diferença efetuada entre o acumulador e o número n8.

Notas:

O resultado de uma subtração se encontra sempre no acumulador.

A subtração entre dois valores de 16 bits não é possível com a instrução SUB.

JR Z JR NZ JR CALL

Todos os programas que estudamos até agora foram concebidos no tipo seqüencial, o que quer dizer que as instruções eram executadas umas após as outras, na ordem em que tinham sido escritas. Todos nós sabemos que em BASIC é possível, com instruções como GOTO por exemplo, impedir que o programa se desenvolva normalmente, obrigando-o a se deslocar a um número de linha escolhido por nós. Vejamos como deveremos agir para obter o mesmo efeito. Encontraremos, após algumas explicações teóricas, o estudo de exemplos bem concretos.

JR Z

Deslocamento se zero

O deslocamento a uma das partes do programa só se fará se uma das duas condições seguintes se realizar:

1. Resultado de uma operação igual a zero.
2. Comparação entre dois números iguais.

É o byte que segue imediatamente a instrução JR Z que, no modo complemento da base dois, indicará então ao microprocessador que outra parte do programa deverá ser executada.

No caso de uma comparação entre dois números diferentes ou de uma operação que não dá zero, JR Z não terá nenhum efeito e é a instrução escrita na linha seguinte que será executada.

Para resumir, digamos que a instrução JR Z se utiliza da mesma maneira que a frase BASIC bem conhecida:

IF A ± B = 0 THEN ...

JR NZ

Deslocamento se não zero

Eis a instrução contrária à precedente. Desta feita, o deslocamento só se efetuará caso ocorra uma das duas situações seguintes.

1. Resultado de uma operação não igual a zero.
2. Comparação efetuada entre dois números diferentes.

Aqui, também, o lugar do programa em que o deslocamento deverá se fazer será indicado pelo byte colocado após a instrução JR NZ. O número escrito no byte deverá sê-lo sob a forma de complemento da base dois.

Pode-se escrever o equivalente BASIC de JR NZ do seguinte modo:

IF A ± B < > 0 THEN ...

JR

Deslocamento em todos os casos

O deslocamento à parte do programa indicado pelo byte que segue a instrução JR é um deslocamento incondicional. Este tipo de deslocamento não se preocupa em saber se esta ou aquela condição foi realizada: ele se efetua necessariamente.

Você reconhece em JR o equivalente assembler do comando BASIC GOTO.

CALL

Deslocamento para um subprograma

Após GOTO, eis GOSUB: CALL é, com efeito, a instrução de deslocamento que permite pular até um subprograma. Trata-se, como para JR, de um deslocamento incondicional que se efetuará em todos os casos.

É inconcebível, em BASIC, escrever um GOSUB sem prever o RETURN que nos levará de volta ao programa principal. O mesmo se

dá em assembler e devemos sempre pensar em terminar os nossos subprogramas com uma instrução que já encontramos desde o nosso primeiro exemplo: RET.

INC

Esta instrução incrementa o conteúdo de um registrador, isto é, ela lhe acrescenta uma unidade. Nós a empregamos no modo de endereçamento registrador.

Exemplo: MODO DE ENDEREÇAMENTO REGISTRADOR (15 bytes)

Programa BASIC

```
10
20
30 SCREEN 0 : Y = USR0 ( X )
```

Programa assembler

Linhas	Códigos de máquina		Assembler	
1	3E 41		LD	A,65
2	06 00		LD	B,0
3	21 00 00		LD	HL,0
4	CD CD 07	ROTINA:	CALL	1997
5	23		INC	HL
6	04		INC	B
7	20 F9		JR	NZ,ROTINA(−7)
8	C9		RET	

Linhas 1, 2 e 3: A inicialização do programa se realiza escrevendo em A o valor 65 e em B e HL o valor 0. O registrador HL contém por enquanto o endereço do primeiro byte da memória do vídeo em modo 0 e A o código ASCII da letra maiúscula A, letra que nos propomos exibir 256 vezes seguidas na tela.

Linha 4: Aqui está o nosso segundo apelo a um subprograma já escrito na memória morta do MSX. Ele realiza o equivalente do co-

mando BASIC VPOKE enviando à memória do vídeo duas indicações:

- Ele lhe indica com precisão, primeiramente, qual o byte que deve ser modificado; transmite o seu endereço via resgistrador HL.
- Indica em seguida que valor deve ser escrito neste byte; fornece este dado via acumulador.

Já que no que se refere ao nosso programa, HL aponta sobre o primeiro byte de tela e já que A contém 65, o caractere cujo código é igual a 65 aparece no alto e à esquerda da imagem: é a letra A. Tudo se passou como se tivéssemos escrito em BASIC: VPOKE 0,65.

Linha 5: Como se acrescenta uma unidade a HL, ele aponta sobre o segundo byte da memória da tela.

Linha 6: O registrador B também é incrementado; o seu novo valor é portanto 1.

Linha 7: Nós nos encontramos agora diante da nossa primeira instrução de deslocamento, por isso vamos com calma.

JR NZ deve se interpretar da seguinte forma: o programa voltará a executar a linha 4 se o resultado da operação precedente for diferente de zero (se é Não Zero, portanto). Ora, qual é a operação da linha 6? Uma adição. E qual é o resultado desta adição? 1, não é? Então o microprocessador recomeçará a executar a linha 4.

Ele retornará à rotina 1997 e uma segunda letra A aparecerá ao lado da precedente. Com efeito, nesta ocasião, HL conterá 1 e o acumulador 65. Depois o computador acrescentará uma unidade a HL e a B. O resultado desta última operação, aqui ainda diferente de 0, ligará de novo a máquina à linha 4. E uma terceira letra será desenhada.

O princípio da rotina deve agora estar compreendido; só falta ver os detalhes:

- O programa pára de executar o ciclo da rotina, quando a instrução INC B da linha 6 der um resultado nulo; isso se produz quando o registrador B, tendo chegado a 255, deve repassar a 0. 256 caracteres A estão então visiveis na tela, dispostos uns ao lado dos outros.
- É preciso se habituar com a maneira em que é apresentada a parte assembler. A linha 4 foi chamada ROTINA e da mesma forma

a linha 7 se escreve JR NZ,ROTINA em vez de se escrever JR NZ, linha 4. JR NZ,ROTINA foi codificado 20 F9. 20 é o código da instrução JR NZ e F9 é, em complemento da base dois, o número −7. Indica-se assim ao microprocessador que ele deve voltar atrás 7 bytes, efetuando-se o desconto sempre a contar do primeiro byte da linha que se segue a instrução de ligação.

0 ⟶ C9 ; −1 ⟶ F9 ; −2 ⟶ 20 ; ...... −7 ⟶ CD

(Sendo CD o primeiro byte da linha à que o deslocamento deve ser feito.)

DIFERENTES FORMAS DE INCREMENTAÇÃO DE UM REGISTRADOR

INC	R	R é um dos registradores A, B, C, D, E, H ou L.
INC	Rd	Rd é um dos registradores BC, DE, HL, SP, IX ou IY.

INC (HL)

Com INC(HL), há a possibilidade de incrementar — acrescentando 1 — o conteúdo de um byte da memória. Utiliza-se o modo de endereçamento indireto

Exemplo: MODO DE ENDEREÇAMENTO INDIRETO (14 bytes)

Programa BASIC

```
10
20
30 INPUT " PRIMEIRO NÚMERO " ; N : POKE 62100 , N
40 INPUT " SEGUNDO NÚMERO " ; M : POKE 62101 , M + 1
50 Y = USRO ( X )
60 PRINT " A SOMA VALE " ; PEEK (62100)
```

Programa assembler

Linhas	Códigos de máquina	Assembler		
1	21 94 F2		LD	HL,62100
2	3A 95 F2		LD	A,(62101)
3	D6 01	ROTINA:	SUB	1
4	28 03		JR	Z,FIM (+3)
5	34		INC	(HL)
6	18 F9		JR	ROTINA (−7)
7	C9	FIM:	RET	

A leitura das linhas BASIC indica que se trata agora de um programa de adição. A instrução ADD não aparece na parte assem-

bler pois foi substituída por uma rotina incrementando o conteúdo de um byte e isso quantas vezes forem necessárias.

Linha 2: O conteúdo do byte 62101 é colocado no acumulador fato que faz com que A contenha o número M + 1. Este valor é igual ao segundo termo da soma, aumentado de 1.

Linha 3: Diminui-se 1 do registrador A. Está assim explicada a razão pela qual partimos de um valor superior a uma unidade por A. Isto compensa aquilo.

Linha 4: Se a diferença incide sobre dois algarismos iguais, isto é, se A vale 1, um deslocamento é efetuado na linha 7, linha chamada FIM. 28 é o código da instrução JR Z e 03 é o número de bytes que o programa terá que saltar. Você se lembra do registrador PC, o contador ordinal? É ele que vai realizar este deslocamento. Ele contém sempre o endereço da próxima instrução a executar; logo, no nosso exemplo, ele é carregado com o número 62011 (faça a conta, o código da instrução INC está realmente neste byte). Assim, quando o teste da linha 4 indicar que uma ligação é necessária, três unidades serão somadas ao registrador PC que irá apontar sobre o byte 62014, byte correspondente a RET. Guarde de tudo isto que o desconto do número de bytes nas operações de deslocamento se faz sempre a partir do início da linha seguinte.

Linha 5: Soma-se 1 ao conteúdo do byte 62100, portanto ao primeiro termo da soma.

Linha 6: JR é a instrução de deslocamento incondicional. Equivalente de GOTO, ela remete o processador à linha 3 para a continuação do programa.

Nós nos encontramos uma vez mais diante de uma rotina. A cada passagem, ela procede às duas seguintes operações:

- 1 é retirado do registrador A, isto é, do segundo número da soma.
- 1 é somado ao byte 62100, isto é, ao primeiro termo da soma.

Desincrementando-se A cada vez, o seu valor chegará forçosamente a 1. A linha assembler 3 efetuará então uma diferença que dá um resultado nulo, e o deslocamento JR Z será realizado. O byte 62100 conterá no fim do programa a soma dos números que propusemos ao computador.

Recapitulemos rapidamente o que já havíamos dito a respeito das

adições de oito bits (ver instrução ADD); quando o resultado de uma soma chega a 256, ele é remetido a zero.

Exemplo:

200 + 100 = 256 + 44 → 44.

DIFERENTES FORMAS DE INCREMENTAÇÃO DE UM BYTE DA MEMÓRIA

INC	(HL)
INC	(IX+n)
INC	(IY+n)
O conteúdo do byte apontado é incrementado.	

PUSH — POP

Estas duas instruções permitem o empilhamento e o desempilhamento de registradores na pilha do sistema. Elas só autorizam o modo de endereçamento registrador.

Exemplo: MODO DE ENDEREÇAMENTO REGISTRADOR (13 bytes)

Programa BASIC

```
10
20
30 CLS : Y = USR0 ( X )
```

Programa assembler

Linhas	Códigos de máquina		Assembler	
1	3E C8		LD	A,200
2	F5	ROTINA:	PUSH	AF
3	3E 46		LD	A,70
4	CD A2 00		CALL	162 (CHPUT)
5	F1		POP	AF
6	3C		INC	A
7	20 F6		JR	NZ,ROTINA(–10)
8	C9		RET	

Linhas 1 e 2: O acumulador é carregado com o valor 200 e este valor é imediatamente protegido na pilha. Pouco importa saber realmente em que byte da memória se deu o empilhamento. O que nos interessa é que poderemos encontrar o valor do acumulador quando tivermos necessidade. Não dê muita atenção ao fato de que a instrução PUSH empilha não somente o registrador A, mas com ele um certo registrador F. Não é absolutamente necessário saber o que é este último, ao menos por enquanto. Saiba simplesmente que

PUSH empilha sempre dois registradores 8 bits e portanto A não lhe é suficiente.

Linha 3: O conteúdo de A é transportado a 70.

Linha 4: Estamos de novo diante da chamada da rotina de exibição. Um primeiro caractere, a letra F (ASCII 70), aparece na tela. Ao mesmo tempo, o cursor é deslocado de um caractere para a direita.

Linha 5: POP é a operação inversa de PUSH. O registrador A encontra então o seu valor inicial e retoma o valor 200.

Linhas 6 e 7: O acumulador, incrementado, passa a 201 e a instrução de ligação remete o processador à linha ROTINA. Por ocasião da segunda execução da rotina, as seguintes ações são efetuadas:

- reserva do número 201 na pilha;
- exibição de uma segunda letra F;
- retomada por A do valor 201;
- incrementação deste registrador.

O programa só pára de rodar quando o acumulador, tendo atingido o máximo 255, é obrigado a repassar em 0. Poderemos ver então no televisor que se deram 56 exibições sucessivas da letra F.

DIFERENTES FORMAS DE UTILIZAÇÃO DA PILHA

PUSH	Rd	
POP	Rd	Rd é um dos pares de registradores AF, BC, DE, HL, IX ou IY.

DEC

O conteúdo do registrador especificado é diminuído, isto é, uma unidade lhe é retirada. O modo de endereçamento registrador é o único utilizável.

Exemplo: MODO DE ENDEREÇAMENTO REGISTRADOR (23 bytes)

Programa BASIC

```
10
20
30 SCREEN 3 : Y = USR0 ( X )
40 GOTO 40
```

Programa assembler

Linhas	Códigos de máquina	Assembler		
1	1E 80		LD	E,128
2	06 1F		LD	B,31
3	0E A8		LD	C,168
4	21 00 00		LD	HL,0
5	78	ROTINA:	LD	A,B
6	CD CD 07		CALL	1997
7	23		INC	HL
8	79		LD	A,C
9	CD CD 07		CALL	1997
10	23		INC	HL
11	1D		DEC	E
12	20 F3		JR	NZ,ROTINA(–13)
13	C9		RET	

Linha 1 a 4: Diversos registradores são inicializados. Sendo o objetivo deste programa intervir na memória da tela, carregamos HL com o endereço do primeiro byte desta memória em modo multicor (SCREEN 3).

Linha 5: O valor 31 é recopiado no acumulador...

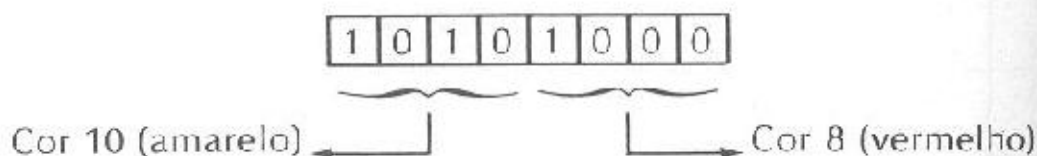
Linha 6: ... e o subprograma de escrita na memória do vídeo é chamado. A sua função é inscrever o valor A no byte da tela apontado por HL. Em BASIC teríamos escrito: VPOKE 0,31.

Neste momento aparece um pequeno retângulo no alto e à esquerda da imagem: ele ocupa a metade superior da superfície de um caractere do tipo SCREEN 1 e é colorido de branco e preto.

Preto	Branco
0001	1111

Byte 0 : 00011111 = 31 decimal

Linhas 7 e 8: Incrementa-se HL, que aponta em conseqüência disso sobre o segundo byte do vídeo. Depois realiza-se uma transferência entre os conteúdos de C e A. O número 168 que se encontrava em C está agora, portanto, também no acumulador.

Linha 9: Uma vez que o processador é relançado no subprograma 1997, um novo retângulo aparece na tela. Situa-se exatamente embaixo do outro e as suas cores são definidas pela decomposição binário do número 168 : 10101000.

Linhas 10 a 12: Faz-se HL apontar sobre o terceiro byte da zona da memória da tela e remete-se o programa à linha 5. Dois novos retângulos se tornarão visíveis, depois dois outros etc.

O computador sairá da rotina ROTINA quando as 128 diminuições sucessivas do registrador E o tiverem levado a 0. 128 quadrados em quadricromia estarão então diante dos nossos olhos.

DIFERENTES FORMAS DE DIMINUIÇÃO DE UM REGISTRADOR

DEC	R	R é um dos registradores A, B, C, D, E, H ou L.
DEC	Rd	Rd é um dos registradores BC, DE, HL, IX ou IY.

DEC (HL)

Esta instrução permite retirar 1 do valor de um byte da memória. Emprega-se o modo de endereçamento indireto.

Exemplo: MODO DE ENDEREÇAMENTO INDIRETO (13 bytes)

Programa BASIC

```
10
20
30 CLS : LOCATE 5,10 : PRINT " INICIO "
40 Y = USR0 ( X ) : LOCATE 5,10 : PRINT " FIM "
```

Programa assembler

Linhas	Códigos de máquina	Assembler		
1	3E FF		LD	A,255
2	21 94 F2		LD	HL,62100
3	77		LD	(HL),A
4	3D	ROTINA:	DEC	A
5	20 FD		JR	NZ,ROTINA (−3)
6	35		DEC	(HL)
7	20 FA		JR	NZ,ROTINA(−6)
8	C9		RET	

Este é o programa que realiza uma rotina de atraso. Ela é equivalente em duração à rotina BASIC:

```
FOR I = 1 TO 250 : NEXT
```

Mas, se o BASIC não fez outra coisa se não contar até 250, o assembler, durante o mesmo tempo, teve o lazer de executar a rotina vazia mais de 60 000 vezes. Isso demonstra a extraordinária rapidez dos programas escritos em linguagem de máquina.

Linhas 1, 2 e 3: O registrador A é carregado com o valor máximo e este valor é escrito no byte 62100.

Linhas 4 e 5: Como A vale 255, a diminuição lhe subtrai uma unidade e o faz passar a 254. Deve-se estar agora na instrução JR NZ que vai remeter o programa à linha 4. Assim, o microprocessador não efetuou nenhuma ação visível: só perdeu tempo. A linha 4 coloca em A o número 253 e de novo a ligação JR NZ faz retornar à linha ROTINA. Logo, encontra-se com estas duas linhas, mas muito mais rápido, a linha BASIC.

```
FOR I = 254 TO 0 STEP - 1 : NEXT
```

Linha 6: Estando o acumulador então em zero, é a vez do byte 62100 ser diminuído. Como havia sido carregado no início com o número 255, vai conter portanto 254.

Linha 7: Nova utilização de JR NZ, que se referirá à última subtração efetuada, no caso a diminuição da linha 6. Uma vez que esta diferença terá sido feita entre os números diferentes 255 e 1, o programa se deslocará de novo para ROTINA, portanto 6 bytes para trás.

Encontra-se então a linha 4. A será mais uma vez diminuído e, partindo de 0, passará de novo a 255. Não esqueçamos de que −1 se escreve para o processador 255 (ou FF hexadecimal). Estamos novamente numa rotina que vai fazer passar A de 255 a 0 (linhas 4 e 5), depois, no fim, uma diminuição do byte 62100 (linha 6) será realizada. Estando ainda este byte no valor 253, haverá ainda ligação na linha 4.

O princípio deve estar compreendido: enquanto o conteúdo do byte 62100 não for nulo, o programa faz o ciclo da rotina. A sua execução terá durado no total menos de meio segundo.

DIFERENTES FORMAS DE DIMINUIÇÃO DE UM BYTE DA MEMÓRIA

DEC	(HL)
DEC	(IX+n)
DEC	(IY+n)
O conteúdo do byte apontado é diminuído.	

DJNZ

É uma instrução de deslocamento condicional: ela diminui o registrador B e só realiza o deslocamento se o conteúdo do registrador não atingiu o valor 0.

Exemplo: MODO DE ENDEREÇAMENTO RELATIVO (24 bytes)

Programa BASIC

```
10
20
30 SCREEN 1 : Y = USR0 ( X )
```

Programa assembler

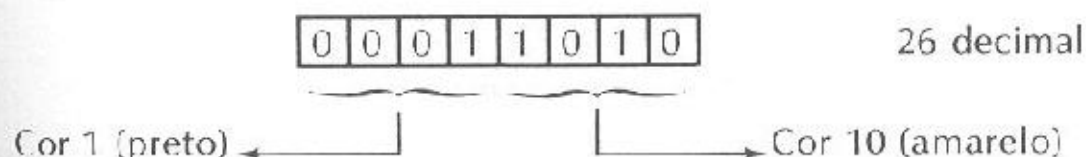
Linhas	Códigos de máquina	Assembler		
1	21 08 20		LD	HL,8200
2	06 04		LD	B,4
3	3E 1A		LD	A,26
4	CD CD 07	ROTINA1:	CALL	1997
5	23		INC	HL
6	10 FA		DJNZ	ROTINA1 (−6)
7	06 04		LD	B,4
8	3E 83		LD	A,131
9	CD CD 07	ROTINA2:	CALL	1997
10	23		INC	HL
11	10 FA		DJNZ	ROTINA2 (−6)
12	C9		RET	

Considerando-se que este programa modifica os valores de alguns bytes da tabela de cores no modo SCREEN 1, paremos alguns minutos para pôr na cabeça esta tabela de 32 bytes.

Números dos bytes	Números dos caracteres	Números dos bytes	Números dos caracteres
8192	0 à 7	8204	96 à 103
....		8205	104 à 111
8200	64 à 71	8206	112 à 119
8201	72 à 79	8207	120 à 127
8202	80 à 87	....	
8203	88 à 95	8221	240 à 255

Linhas 1 e 2: Faz-se HL apontar para o byte 8200 da zona do vídeo e escreve-se em B o valor 4. B nos servirá de contador numa rotina que vamos executar quatro vezes.

Linhas 3 e 4: Após ter carregado o acumulador com o número 26, lança-se o processador no subprograma de escrita da memória da tela. O conteúdo do byte 8200 então é levado a 26. Teríamos obtido em BASIC o mesmo resultado com VPOKE 8200,26.

Decomponhamos 26 em binário para ter uma idéia das cores que os nossos caracteres vão tomar:

Só falta indicar de que caracteres o byte 8200 se ocupa: são as letras @, A, B, C, ... G.

Linhas 5 e 6: Acrescenta-se uma unidade a HL e executa-se DJNZ. Esta instrução é perfeitamente equivalente ao par:

```
DEC  B
JR   NZ,ROTINA1
```

O programa retorna então à rotina ROTINA1 e escreve o valor 26 no byte 8201. Este mesmo valor será em seguida escrito nos bytes 8202 e 8203. A partir deste momento, digitando-se uma das teclas cujo código está incluído no intervalo 64 — 95 (aproximadamente o alfabeto maiúsculo) aparecerá na tela um caractere colorido de preto sobre amarelo.

Linhas 7 a 12: Desta vez nenhuma indicação é dada. Deixamos ao próprio leitor o cuidado de determinar o papel da segunda parte do programa.

Nota: Não existe instrução equivalente a DJNZ para os outros registradores.

JR C JR NC

Abordamos agora duas novas instruções de deslocamento que vão poder, quando as condições desejadas se realizarem, pôr um novo valor no registrador PC e permitir assim anular o desenvolvimento seqüencial do programa. Estas instruções vão incidir na comparação das grandezas de dois números, cujo primeiro estará sempre em um registrador. Lembremos que para poder utilizar JR Z e JR NZ, será preciso que uma operação ou uma comparação tenha sido efetuada anteriormente. O mesmo se dará para estas duas novas instruções.

JR NC

Deslocamento se superior ou igual

JR NC realizará um deslocamento num dos bytes do programa de máquina, num dos seguintes casos:

1. Subtração entre dois números cujo primeiro é superior ou igual ao segundo.
2. Comparação entre dois números cujo primeiro, contido no acumulador, é superior ou igual ao segundo.

O byte que segue a instrução JR NC indica com precisão, no modo do complemento da base dois, que parte do programa deverá então ser executada. Naturalmente esta instrução não terá efeito se o primeiro número for inferior ao outro.

Observe-se que a comparação se fará sem que o computador leve em consideração o valor do complemento da base dois destes números. Se, por exemplo, os dois números valem 254 e 10, o primeiro será considerado como superior ao segundo, se bem que seja em realidade o número −2 no complemento da base dois.

Se quisermos encontrar o equivalente BASIC de JR NC, devemos escrever:

IF A > =B THEN ...

JR C

Deslocamento se inferior

Desta vez, o deslocamento efetuar-se-á num dos seguintes casos:

1. Subtração entre dois números cujo primeiro é estritamente inferior ao segundo.
2. Comparação entre dois números cujo primeiro é estritamente inferior ao segundo.

Por ocasião de uma comparação, o primeiro número está sempre contido no acumulador.

Para esta instrução também, o computador não levará em conta valoes negativos, os que correspondem ao complemento da base dois. Escrevamos a linha BASIC correspondente:

IF A <B THEN ...

CP

Uma comparação é realizada entre o acumulador e o número descrito imediatamente após esta instrução. Um comando de deslocamento deve seguir normalmente esta comparação. Os modos de endereçamento imediato, registrador e indireto podem ser utilizados.

Exemplo: MODO DE ENDEREÇAMENTO INDIRETO (25 bytes)

Programa BASIC

```
10
20
30 X = INT ( RND(1) * 256 ) : POKE 62100 , X
40 INPUT " QUE NÚMERO VOCÊ PROPÕE " ; N :
   POKE 62101 , N
50 Y = USR0 ( X ) : ON PEEK ( 62102 ) GOTO 60 , 70 , 80
60 PRINT " VOCÊ GANHOU " : END
70 PRINT " GRANDE DEMAIS " : GOTO 40
80 PRINT " PEQUENO DEMAIS " : GOTO 40
```

Programa assembler

Esta é uma versão do jogo que consiste em adivinhar um número que o computador terá escolhido. O deslocamento ON GOTO da linha BASIC 50 se efetuará em função do número achado no byte 62102. Vejamos como o assembler terá colocado nele o valor correto 1, 2 ou 3.

Linhas	Códigos de máquina			
		Assembler		
1	21 94 F2		LD	HL,62100
2	3A 95 F2		LD	A,(62101)
3	BE		CP	(HL)
4	28 06		JR	Z,IGUAL (+6)
5	30 08		JR	NC,SUP (+8)
6	3E 03	INF:	LD	A,3
7	18 06		JR	FIM (+6)
8	3E 01	IGUAL:	LD	A,1
9	18 02		JR	FIM (+2)
10	3E 02	SUP:	LD	A,2
11	32 96 F2	FIM:	LD	(62102),A
12	C9		RET	

Linha 2: A linha BASIC 40 terá escrito no byte 62101 o número N proposto. Logo é o registrador A que vai conter este número.

Linha 3: Uma comparação é efetuada entre o número proposto e o conteúdo do byte 62100. Ora, neste byte foi inscrito por POKE o número X que o computador tirou por acaso. Esta é portanto a linha que vai realizar a comparação na qual é baseado todo o programa.

Linha 4: Se a comparação incidiu em dois números iguais, isto indicará que ganhamos. JR Z vai proceder então a um deslocamento para a linha 8, seis bytes mais longe. LD A,1 vai, neste momento, colocar no acumulador o número 1 e um deslocamento incondicional (linha 9) vai levar o processador à antepenúltima linha do programa. Só faltará então escrever o valor 1 no byte 62102. O BASIC encontrará este número e a instrução ON GOTO fará imprimir a mensagem "Você ganhou".

Linha 5: Se supõe agora que A é superior ao número escolhido pelo computador, a instrução JR NC nos conduzirá à linha 10. O registrador A será carregado com o valor 2, valor que será em seguida escrito (linha 11) no byte 62102. Bastará que o BASIC encontre o conteúdo deste byte e a mensagem "GRANDE DEMAIS" será mostrada na tela.

Linhas 6 e 7: Finalmente, última possibilidade, propusemos à máquina um número pequeno. As instruções JR Z e JR NC ficaram sem

efeito e o programa desenvolveu-se seqüencialmente até estas linhas. O algarismo 3 será escrito no acumulador antes que um deslocamento incondicional envie o microprocessador à linha 11. 3 é então recopiado no byte 62102 e a instrução BASIC ON GOTO dará a resposta à nossa experiência. "PEQUENO DEMAIS".

DIFERENTES FORMAS DA INSTRUÇÃO DE COMPARAÇÃO

CP	n8	Comparação entre A e um valor de 8 bits.
CP	R	Comparação entre A e um dos registradores de 8 bits R.
CP CP CP	(HL) (IX + n) (IY + n)	Comparação entre A e conteúdo de um byte de memória.

OR

Um OU lógico é efetuado entre o acumulador A de um lado e um número de 8 bits, ou um registrador simples, ou o conteúdo de um byte de outo lado. Os modos de endereçamento imediato, registrador e indireto são portanto permitidos.

Exemplo: MODO DE ENDEREÇAMENTO IMEDIATO (19 bytes)

Programa BASIC

```
10
20
30 INPUT " DÊ UM NÚMERO " ; N
40 POKE 62100,N : Y = USR0 ( X )
50 ON PEEK ( 62101 ) GOTO 60 , 70
60 PRINT " O NÚMERO É IMPAR " : GOTO 30
70 PRINT " O NÚMERO É PAR " : GOTO 30
```

Programa assembler

Linhas	Códigos de máquina	Assembler		
1	21 94 F2		LD	HL,62100
2	7E		LD	A,(HL)
3	F6 01		OR	1
4	BE		CP	(HL)
5	20 04		JR	NZ,PAR (+4)
6	3E 01		LD	A,1
7	18 02		JR	FIM (+2)
8	3E 02	PAR:	LD	A,2
9	32 95 F2	FIM:	LD	(62101),A
10	C9		RET	

Reservemos algumas linhas para rever de que forma se executa um OU lógico entre dois números:

```
     1100
  OU 1010
  -------
  =  1110
```

Nosso programa, por seu lado, vai efetuar um OU entre o conteúdo do registrador A e o número 1. Uma vez que 1 se escreve em binário 00000001, somente o último bit de A estará em jogo. Assim, se A se termina por 0, ele será modificado pois o seu último algarismo passará a 1. Ao contrário, se o último algarismo vale 1, A conservará o mesmo valor.

Linhas 1 e 2: O número que se propôs ao computador foi colocado por POKE no byte 62100, logo, o registrador A é carregado com este valor.

Linha 3: O OU lógico é realizado entre o número que escolhemos e a unidade. Se este número for par, a sua escrita binária se fará com um 0 no fim, e se for ímpar, ele terminará por 1. A ação de OR vai portanto consistir em modificar o valor do nosso registro unicamente quando ele for par.

Linhas 4 e 5: A comparação é feita entre os conteúdos do acumulador e do byte 62100, byte que, não esqueçamos, contém o número que digitamos. Quando este número for par, OR o transforma e um deslocamento à linha 8 se efetua.

Linhas 6 e 7: Quando se trata de um número ímpar, o valor 1 é posto em A para ser reescrito em seguida (linha 9) no byte 62101.

Linha 8: Se não, é o número 2 que vai então transitar pelo acumulador para ser colocado neste mesmo byte.

A linha BASIC 50 vai então examinar este byte e a ligação ON GOTO enviará, neste momento, ao computador a instrução correta.

DIFERENTES FORMAS DO OU LÓGICO

OR	n8	n8 é um número de 8 bits.			
OR	R	R é um dos registradores simples.			
OR	(HL)	OR	(IX+n)	OR	(IY+n)

AND

Um E lógico é realizado entre o acumulador e um valor de 8 bits tomado como um dado, um conteúdo do registrador ou um conteúdo da memória. Podemos utilizar os modos de endereçamento imediato, registrador ou indireto.

Exemplo: MODO DE ENDEREÇAMENTO IMEDIATO (15 bytes)

Programa BASIC

```
10
20
30 Y = USR0 ( X )
```

Programa assembler

Linhas	Códigos de máquina		Assembler	
1	3E 5A		LD	A,90
2	CD A2 00	ROTINA:	CALL	162
3	3D		DEC	A
4	E6 FE		AND	254
5	FE 41		CP	65
6	30 F6		JR	NC,ROTINA(–10)
7	C9		RET	

Linha 1: Inicializa-se o acumulador com o valor 90; este número é o código do caractere que se quer fazer escrever na imagem pela rotina 162.

Linha 2: A letra Z, cujo código ASCII é 90, é exibida na tela.

Linhas 3 e 4: A instrução de diminuição substitui 90 por 89 e o computador executa a nossa nova instrução realizando um E lógico entre os números 89 e 254. Vejamos a nível de binário:

```
   0 1 0 1 1 0 0 1 <———— 89
 E 1 1 1 1 1 1 1 0 <———— 254
 ------------------
 = 0 1 0 1 1 0 0 0 <———— 88
```

Reencontramos a particularidade já mencionada do comando AND, que permite forçar a 0 qualquer algarismo binário de um número. No nosso caso, foi o último deles que passou a 0.

Linha 5: Possuindo o acumulador um valor superior a 65, o programa reencontra a linha ROTINA. Uma segunda letra é então impressa no televisor, exatamente ao lado da precedente. O seu código ASCII é igual a 88; trata-se, portanto, da letra X.

Continuemos a acompanhar o trabalho do microprocessador:

DEC A : o registrador A passa a 87 (ou seja 01010111 binário).
AND 254 : este mesmo registrador perde o seu último algarismo 1 e vale então 86 (01010110 binário).

Quando o subprograma de exibição for chamado, é a letra V que vai aparecer desta vez. É inútil ir mais adiante. Você compreendeu que este subprograma escreve o alfabeto em ordem decrescente e saltando uma em cada duas letras. A última delas será a letra B (ASCII 66).

DIFERENTES FORMAS DO E LÓGICO

AND n8	n8 é um número de 8 bits.	
AND R	R é um dos registradores simples.	
AND (HL)	AND (IX+n)	AND (IY+n)

XOR

Como as duas instruções precedentes estudadas, XOR realiza uma operação lógica: um OU exclusivo é efetuado entre o registrador A e um número de 8 bits. Aqui pode-se também utilizar os modos de endereçamento imediato, registrador e indireto.

Exemplo: MODO DE ENDEREÇAMENTO IMEDIATO (16 bytes)

Programa BASIC

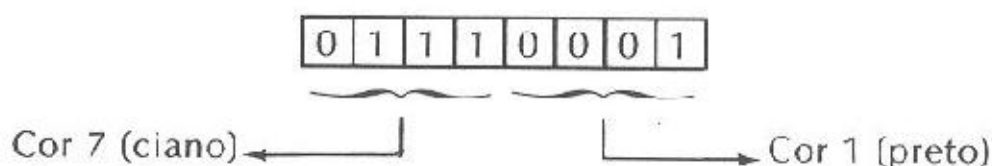
```
10
20
30 SCREEN 3 : Y = USR0 ( X )
40 GOTO 40
```

Programa assembler

Linhas	Códigos de máquina		Assembler	
1	06 00		LD	B,0
2	3E 71		LD	A,113
3	21 00 00		LD	HL,0
4	CD CD 07	ROTINA:	CALL	1997
5	23		INC	HL
6	EE FF		XOR	255
7	10 F8		DJNZ	ROTINA (−8)
8	C9		RET	

Linhas 1 a 3: Antes de pedir ao computador que efetue para nós uma escrita na memória da tela, é necessário inicializar o programa.

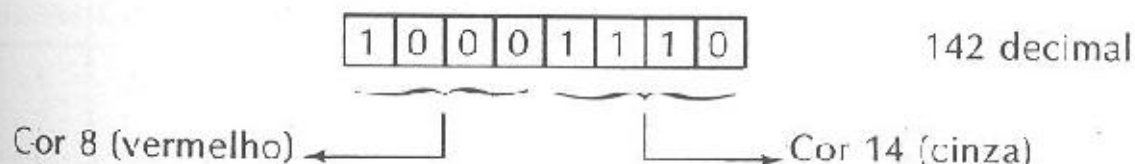
Linha 4: No modo SCREEN 3, a tela é constituída de 1 536 retângulos elementares cujas partes da esquerda e da direita podemos colorir como quisermos.

Para saber como vai aparecer o primeiro destes retângulos, devemos decompor em binário o valor 113:

O primeiro retângulo é portanto constituído de um quadrado azul-claro e de um outro preto.

Linha 6: A utilização do OU exclusivo é feita aqui com o objetivo de inverter todos os algarismos de um número binário. Com efeito, quando XOR é efetuado entre um algarismo e 1, este algarismo muda de valor.

Para nosso exemplo, segue-se:

```
      0 1 1 1 0 0 0 1  <-------  113 (registrador A)
XOR   1 1 1 1 1 1 1 1  <-------  255
---------------------
=     1 0 0 0 1 1 1 0  <-------  142 (registrador A)
```

Linha 7: O conteúdo do registrador B é diminuído de uma unidade e toma o valor 255. O programa se desloca de novo para a linha 4. Como HL foi incrementado e o acumulador vale 142, somos levados á seguinte afirmação: um segundo retângulo vai aparecer abaixo do precedente e será colorido de vermelho e cinza.

Só temos que analisar o que proporciona uma nova utilização de XOR.

142 XOR 255 = 113

A encontra então o seu valor de origem. É por isso que o nosso programa nos mostra, no fim da execução, 256 retângulos coloridos alternativamente de azul e preto, depois de vermelho e cinza.

DIFERENTES FORMAS DO **OU** EXCLUSIVO

XOR	n8	n8 é um número de 8 bits.			
XOR	R	R é um dos registradores simples.			
XOR	(HL)	XOR	(IX+n)	XOR	(IY+n)

SRL A

Abreviação de Shift Right Logical, esta instrução desloca todos os bits do acumulador A para a direita. O bit da esquerda é posto em zero. Trata-se do modo de endereçamento registrador.

Exemplo: MODO DE ENDEREÇAMENTO REGISTRADOR (9 bytes)

Programa BASIC

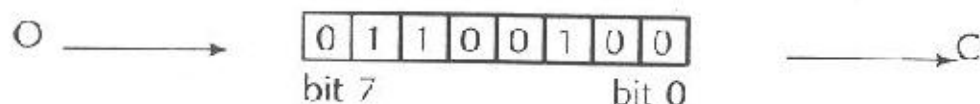
```
10
20
30 PRINT " DÊ UM NÚMERO MENOR QUE 256 " ;
40 INPUT N : POKE 62100 , N : Y = USR0 ( X )
50 PRINT " O QUOCIENTE INTEIRO DO NÚMERO " ;
60 PRINT " POR DOIS VALE " ; PEEK (62101) : GOTO 30
```

Programa assembler

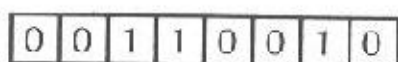
Linhas	Códigos de máquina	Assembler	
1	3A 94 F2	LD	A,(62100)
2	CB 3F	SRL	A
3	32 95 F2	LD	(62101),A
4	C9	RET	

Este programa efetua em assembler a divisão inteira de um número por dois. Vejamos, a nível de binário, como isso se passa.

Consideremos o número decimal 100 que se escreve em binário 01100100.

REGISTRADOR A

Façamos os algarismos que constituem este número sofrerem um deslocamento para a direita. Cada algarismo vai se encontrar no bit de fila imediatamente inferior: este algarismo do bit 7 (0) vai passar ao bit 6; o algarismo do bit 6 vai se escrever no bit 5 etc. O último algarismo à direita (bit 0) vai, portanto, sair do byte e estará perdido para nós. O computador, entretanto, guardará a identificação, levando-o a um lugar especial que se chama o indicador de reserva e que se anota com C. Este indicador tomará, portanto, o valor 0, mas insistindo, isso não tem nenhuma importância para nosso exemplo.

Sabendo que SRL substitui sempre o bit 7 por 0, obtém-se então o seguinte para o registrador A:

0	0	1	1	0	0	1	0

A tradução em decimal deste valor é 50; logo, dividimos realmente o número 100 por 2.

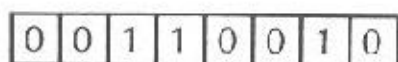
E se tivéssemos partido de um número ímpar? Experimentemos com 101.

O ——→

0	1	1	0	0	1	0	1

——→ C

Quando SRL tiver agido, obteremos.

0	0	1	1	0	0	1	0

isto é, 50, o que é exatamente o quociente inteiro de 101 por 2. No segundo caso, o indicador de reserva passou a 1.

Só falta compreender porque o deslocamento para a direita dos algarismos conduziu a uma divisão por dois. Tomemos por exemplo o algarismo 1 do bit 6 e vejamos o que ele vai valer: ele valia 2^6, isto é, 64, antes de SRL, ele vale 2^5, ou seja 32, depois; logo ele foi reduzido à metade. Este mesmo raciocínio se faz para todos os outros algarismos, o que nos dá a explicação.

As diferentes linhas não serão estudadas um por um, desta vez, pois o programa assembler é compreendido sem dificuldade.

DIFERENTES FORMAS DA ROTAÇÃO LÓGICA PARA A DIREITA (MODO REGISTRADOR)

SRL	R	R é um dos registradores de 8 bits.

SRL (IX + n)

Como para SRL A, é um deslocamento para a direita, mas este deslocamento refere-se ao conteúdo de um byte de memória. O modo de endereçamento permitido é portanto o indireto.

Exemplo: MODO DE ENDEREÇAMENTO INDIRETO (29 bytes)

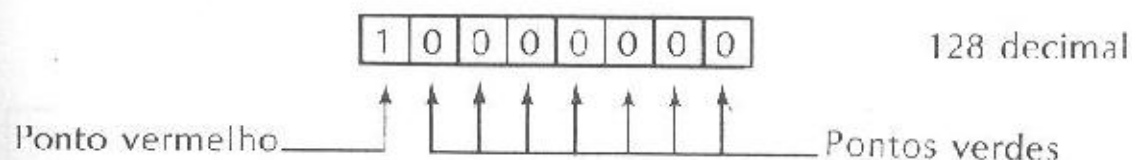
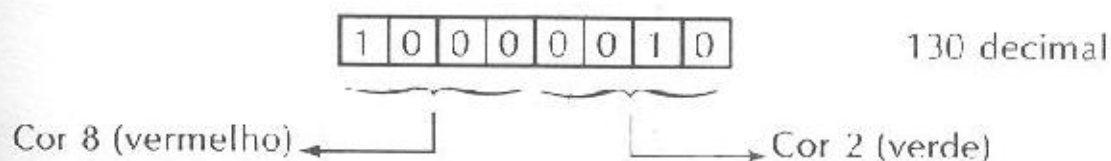
Programa BASIC

```
10
20
30 SCREEN 2 : FOR I = 0 TO 255
40 VPOKE I , 128 : VPOKE I + 8192 , 130 : NEXT
50 FOR I = 1 TO 8 : Y = USR0 ( X )
60 FOR J = 1 TO 500 : NEXT : NEXT
```

Programa assembler

Linhas	Códigos de máquina	Assembler		
1	DD 21 94 F2		LD	IX,62100
2	06 00		LD	B,0
3	21 00 00		LD	HL,0
4	CD D7 07	ROTINA:	CALL	2007 (RDVRM)
5	DD 77 00		LD	(IX+0),A
6	DD CB 00 3E		SRL	(IX+0)
7	DD 7E 00		LD	A,(IX+0)
8	CD CD 07		CALL	1997
9	23		INC	HL
10	10 ED		DJNZ	ROTINA (−19)
11	C9		RET	

O BASIC escreve o valor 128 em cada um dos 256 primeiros bytes da memória da tela no modo SCREEN 2. Ao mesmo tempo, leva ao nível 130 todos os bytes correspondentes da tabela de cores.

Por conseguinte, veremos aparecer na tela 256 segmentos cujo ponto de esquerda será visível em vermelho e os outros sete pontos em verde.

As cores vermelho e verde são deduzidas da estrutura do número 130.

Terminado o seu trabalho, o BASIC chama o programa assembler uma primeira vez.

Linha 4: É o primeiro exemplo no qual se encontra o subprograma 2007. É muito útil porque ele coloca no acumulador o conteúdo do byte do vídeo indicado por HL. O registrador A se encontra de novo carregado com o valor inscrito no byte de endereço 0, isto é, com 128. CALL 2007 é portanto equivalente a A = VPEEK (HL).

Linhas 5 a 7: Transfere-se o conteúdo do acumulador para o byte 62100, depois dá-se um desvio para a direita a cada um dos bits deste byte:

Antes de SRL : 10000000 (128 decimal)
Após SRL : 01000000 (64 decimal)

O número 64 é em seguida carregado em A. Você teria razão de pensar que as linhas 5 a 7 poderiam ser substituídas pelo único comando do SRL A. Elas só estão aí, com efeito, para lhe mostrar como se utiliza SRL no modo indireto.

Linha 8: A rotina 1997 escreve no byte da RAM do vídeo, indicado por HL, o valor do acumulador. O primeiro segmento da tela nos aparece agora com o seu segundo ponto iluminado em vermelho e todos os outros em verde. Os 255 seguintes sofrerão a mesma transformação.

O BASIC deixará então correr um pouco de tempo e o computador voltará a executar o program assembler uma segunda vez, uma terceira, ... uma oitava. Cada vez, os pontos vermelhos dos segmentos serão deslocados para a direita e acabarão, aliás, desaparecendo por completo.

DIFERENTES FORMAS DA ROTAÇÃO LÓGICA PARA A DIREITA (MODO INDIRETO)

SRL (HL)	SRL (IX+n)	SRL (IY+n)

SLA A

Trata-se desta vez de um deslocamento à esquerda, sendo SLA a abreviação de Shift Left Arithmetic. *O bit 7 passa no indicador de reserva e o bit 0 do registrador A é posto em zero. SLA é utilizado aqui com o modo de endereçamento registrador.*

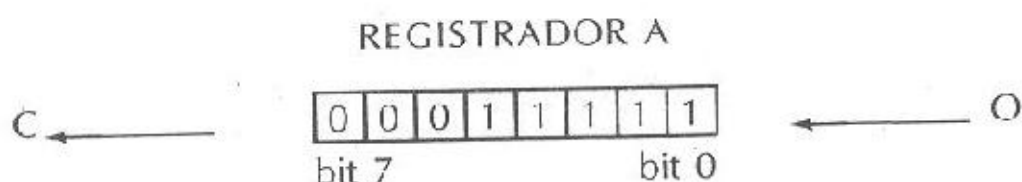
Exemplo: MODO DE ENDEREÇAMENTO REGISTRADOR (9 bytes)

Programa BASIC

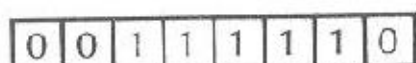
```
10
20
30 INPUT " DÊ UM NÚMERO " ; N : POKE 62100 , N
40 J = 1 : FOR I = 1 TO 3
50 J = J * 2 : Y = USR0 ( X )
60 PRINT " O PRODUTO DO NÚMERO POR " ; J ;
70 PRINT " VALE " ; PEEK ( 62100 )
80 NEXT : GOTO 30
```

Programa assembler

Linhas	Códigos de máquina	Assembler
1	3A 94 F2	LD A,(62100)
2	CB 27	SLA A
3	32 94 F2	LD (62100),A
4	C9	RET

Vamos supor que se propõe ao computador o número 31 e ver no que ele se transforma quando se executa a instrução SLA. 31 tem por equivalente binário 00011111.

REGISTRADOR A

O deslocamento para a esquerda vai fazer sair do registrador o conteúdo do bit 7 que irá se colocar no indicador de reserva, indicador cuja importância é nula no grau de adiantamento dos nossos conhecimentos. Estando todos os algarismos transladados, obtém-se para A:

0	0	1	1	1	1	1	0

O número 0 veio tomar o lugar deixado livre no bit 0. Transcrevendo o resultado obtido em decimal, obtém-se o número 62, isto é, o dobro de 31. Assim o deslocamento de todos os bits para a esquerda permitiu efetuar o produto por 2 do número que se encontrava no acumulador. Isso se pode compreender porque, em definitivo, cada algarismo se encontrará com uma potência de 2 superior de uma unidade à precedente.

Voltemos ao nosso programa: o número que se deu na partida ao computador é colocado no byte 62100, e na primeira passagem da rotina FOR NEXT, este número é multiplicado por 2. A linha assembler 3 recoloca a resposta neste mesmo byte 62100. Na segunda passagem, o número é de novo multiplicado por 2, o que faz com que o valor do início seja, desta vez, multiplicado por 4; ele o será por 8 quando o programa BASIC estiver terminado.

Claro, este programa dá respostas coerentes enquanto não se escolher um número superior ou igual a 32 (ou seja, 00100000 em binário). A partir deste valor, com efeito, é um algarismo 1 que é perdido nos deslocamentos, tornando o resultado final incorreto (mas explicável).

DIFERENTES FORMAS DA INSTRUÇÃO SLA

SLA	R	R é um dos registradores de 8 bits.

SLA (IX + n)

Um deslocamento de um bit para a esquèrda do conteúdo de um byte da memória é efetuado. O bit 0 passa a 0 e o bit 7 é escrito no indicador de reserva.

Exemplo: MODO DE ENDERECAMENTO INDIRETO (28 bytes)

Programa BASIC

```
10
20
30 COLOR , 14 : SCREEN 2 : POKE 62100 , 1 : Y = USR0 ( X )
40 GOTO 40
```

Programa assembler

Linhas	Códigos de máquina		Assembler	
1	06 08		LD	B,8
2	DD 21 94 F2		LD	IX,62100
3	21 00 00	INÍCIO:	LD	HL,0
4	DD 7E 00	ROTINA:	LD	A,(IX+0)
5	CD CD 07		CALL	1997
6	23		INC	HL
7	7C		LD	A,H
8	FE 18		CP	24
9	20 F4		JR	NZ,ROTINA (−12)
10	DD CB 00 26		SLA	(IX+0)
11	10 EB		DJNZ	INÍCIO (−21)
12	C9		RET	

*Linha 2:*O registrador IX é carregado com o endereço de um byte ao qual se vão aplicar 8 deslocamentos sucessivos para a esquerda. A configuração binária do byte 62100 é agora a seguinte: 00000001 (instrução POKE da linha BASIC 30).

Linhas 3 a 5: O subprograma de escrita na RAM do vídeo é ativado; o primeiro segmento da tela aparece então na imagem. O seu ponto da direita é azul e os seus outros sete pontos cinza (a cor de fundo com efeito, é o cinza, neste exemplo).

Linhas 6 a 9: Propõe-se renovar a operação com todos os outros segmentos da memória da tela. Basta portanto fazer com que o acumulador guarde o mesmo valor, o do byte 62100 neste caso, e que HL aponte sucessivamente para cada byte do intervalo 0, 1, 2 ... 6143. Todo o problema consiste em fazer o programador reconhecer que HL chegou a 6144 e que não há mais necessidade de fazer o ciclo da rotina nas linhas ROTINA.

Olhemos as decomposições em peso forte e peso fraco dos números 6143 e 6144.

6143 = 23 * 256 + 255, ou seja, HL = 6143 ; H = 23 ; L = 255;
6144 = 24 * 256 + 0, ou seja, HL = 6144 ; H = 24 ; L = 0

A conclusão é sem ambigüidade: para saber se HL atingiu 6144, basta saber se H atingiu 24. É para isso que serve a instrução CP 24 colocada na linha 8. Você observará que, como esta instrução só se emprega com o acumulador, fomos obrigados a transferir o conteúdo do registrador H para o registrador A.

Linha 10: Quando o computador chegar a esta linha, ela já fez aparecer 6 144 segmentos em azul sobre fundo cinza. Uma vez que somente o último ponto destes segmentos é colorido, pode-se por conseguinte ver no televisor 32 linhas verticais azuis.

Aí o byte 62100 sofre a sua primeira rotação para a esquerda; partindo de 00000001, ele passa a 00000010. Em seguida o programa retoma a sua ação na linha INÍCIO. Os nossos 6 144 bytes vão, uns após os outros, receber o valor binário 00000010 e o segundo ponto partindo da direita de cada um deles se tornará visível. As 32 linhas que aparecerão na tela serão então deslocadas em relação às 32 linhas precedentes, de um pixel para a esquerda.

Pois bem, você deve ter compreendido agora porque as linhas verticais davam, a cada passagem na rotina INÍCIO, a impressão de se deslocarem para a esquerda da imagem.

DIFERENTES FORMAS DA INSTRUÇÃO SLA
(MODO INDIRETO)

SLA (HL)	SLA (IX+n)	SLA (IY+n)

RL A

Todos os bits do acumulador sofrem uma rotação para a esquerda. O bit 7 passa no indicador de reserva e o valor previamente contido por este é transferido para o bit 0. RLA é a abreviação de Rotate Left A.

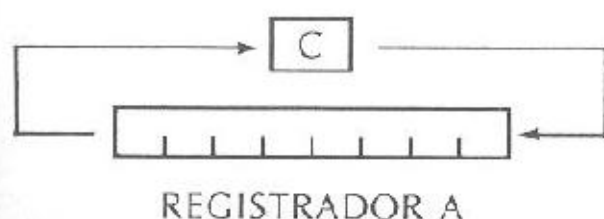
Exemplo: MODO DE ENDEREÇAMENTO REGISTRADOR (24 bytes)

Programa BASIC

```
10
20
30 INPUT " DÊ UM NÚMERO " ; N : POKE 62100 , N
40 Y = USR0 ( X ) : PRINT "SEU DOBRO VALE " ;
50 PRINT 256 * PEEK (62101) + PEEK (62102) : GOTO 30
```

Programa assembler

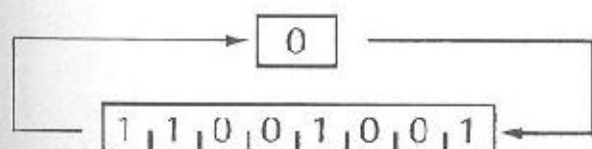
Linhas	Códigos de máquina	Assembler	
1	DD 21 94 F2	LD	IX,62100
2	3E 00	LD	A,0
3	06 00	LD	B,0
4	CB 17	RL	A
5	DD 7E 00	LD	A,(IX+0)
6	CB 17	RL	A
7	CB 10	RL	B
8	DD 70 01	LD	(IX+1),B
9	DD 77 02	LD	(IX+2),A
10	C9	RET	

Você se lembra da instrução SLA? Ela nos permitia multiplicar um número por 2, 4 ou 8 mas isso não ocorria sem algum transtorno grave: os algarismos 1 que saíam na esquerda do acumulador se perdiam e se começávamos de um número grande demais, a resposta não era a esperada. Vejamos como poderemos remediar isso com a nossa instrução RL A.

REGISTRADOR A

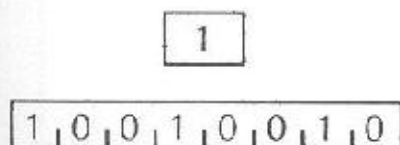
Todos os bits do acumulador sofrem um deslocamento para a esquerda, o bit contido no indicador de reserva passa para o bit 0 e é o bit 7 que toma o seu lugar. Trata-se portanto de uma rotação realizada em 9 bits.

Linhas 2 e 3: Os dois registradores de 8 bits são colocados em zero.

Linha 4: Faz-se A sofrer uma rotação; uma vez que A se escreve 00000000 em binário, isso não tem outro efeito senão o de pôr o algarismo 0 no indicador de reserva.

Linhas 5 e 6: Recopia-se em A o número que escrevemos por POKE no byte 62100 e, graças a RL A, nós o multiplicamos por 2. Examinemos de mais perto e suponhamos, para fixar as idéias, que N tenha sido escolhido como igual a 201 (ou seja, 11001001).

C está em zero (linha 4) e obtemos portanto após RL A:

1

1 0 0 1 0 0 1 0

O bit de C passou a 1 e o nosso valor do acumulador é, em decimal, 146. Isso não é naturalmente o dobro de 201, mas esperemos a continuação.

Linhas 7 e 8: RL B tem por efeito deslocar os 8 algarismos 0 do registrador B e fazer entrar à direita o bit que se encontrava no indicador, isto é, o bit 1. O novo valor de B é portanto 1; ele é inscrito então no byte 62101.

Linhas 9 e 10: O decimal 146 é, por sua vez, colocado no endereço 62102 e o retorno ao BASIC é programado.

Verifiquemos a lógica do programa assembler:

PRINT 256 * PEEK (62101) + PEEK (62102)

Resposta : 256 * 1 + 146 = 402

Logo, tudo se passou definitivamente como se tivéssemos feito um deslocamento de 9 bits.

011001001 (201 decimal) se tornaria
110010010 (402 decimal)

DIFERENTES FORMAS DA ROTAÇÃO PARA A ESQUERDA (MODO REGISTRADOR)

RL	R	R é um dos registradores de 8 bits.

RL (IY + n)

Todos os bits do byte da memória especificado são deslocados de uma posição para a esquerda. O bit 7 é colocado no indicador de reserva e o valor de origem deste é transferido para o bit 0.

Exemplo: MODO DE ENDEREÇAMENTO INDIRETO (28 bytes)

Programa BASIC

```
10
20
30 SCREEN 3 : Y = USR0 ( X )
40 GOTO 40
```

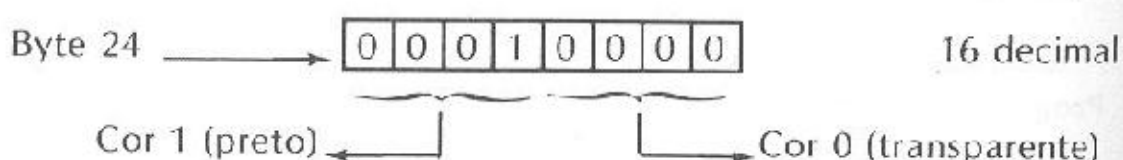
Programa assembler

Linhas	Códigos de máquina	Assembler		
1	06 08		LD	B,8
2	21 18 00		LD	HL,24
3	FD 21 94 F2		LD	IY,62100
4	FD 36 00 10		LD	(IY+0),16
5	FD 7E 00	ROTINA:	LD	A,(IY+0)
6	CD CD 07		CALL	1997
7	23		INC	HL
8	37		SCF	
9	FD CB 00 16		RL	(IY+0)
10	10 F2		DJNZ	ROTINA (−14)
11	C9		RET	

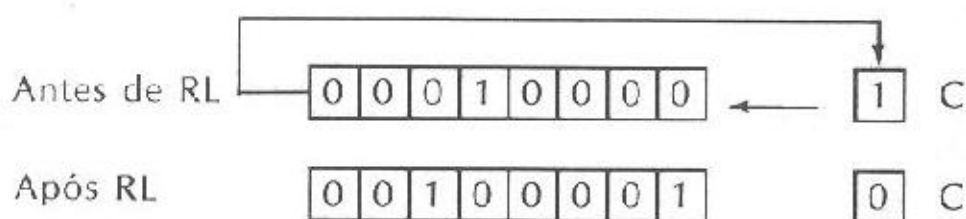
Linhas 1 e 2: 8 bytes da memória da tela vão ser abrangidos por este programa. O modo escolhido, SCREEN 3, e o valor de partida

de HL nos indicam que estes bytes vão corresponder a 8 retângulos colocados uns embaixo dos outros na parte superior esquerda do televisor.

Linhas 3 e 4: O byte 62100 vai servir para acionar a nossa nova instrução. Por exemplo, ele contém o valor 16, ou seja, 00010000 binário.

Linhas 5 e 6: O número 16 é colocado no acumulador e o computador é deslocado para o subprograma 1997. A ação desta rotina, que agora dominamos bem, vai constar em escrever o valor 16 no byte do vídeo 24. O retângulo elementar que é ligado a este byte aparece portanto nas cores preta e azul.

Linha 7: HL, incrementado, aponta para o byte nº 25.

Linha 8: A instrução SCF tem uma função bem precisa: ela força o bit de reserva C a valer 1.

Linha 9: O byte 62100 é submetido a uma rotação para a esquerda. O seu bit 7, que vale 0, passa para o indicador de reserva e o seu bit 0 é substituido pelo antigo conteúdo de C, isto é, por 1.

Linha 10: O retorno à rotina ROTINA será programado enquanto o registrador B não valer 0. Quando isso acontecer, 8 retângulos dispostos uns embaixo dos outros poderão ser vistos na tela; eles terão então as seguintes cores:

Byte 24	: 00010000	preto e azul
Byte 25	: 00100001	verde e preto
Byte 26	: 01000011	azul e verde
Byte 27	: 10000111	vermelho e ciano
Byte 28	: 00001111	azul e branco
Byte 29	: 00011111	preto e branco
Byte 30	: 00111111	verde e branco
Byte 31	: 01111111	ciano e branco

DIFERENTES FORMAS DA ROTAÇÃO PARA A ESQUERDA (MODO INDIRETO)

RL (HL)	RL (IX+n)	RL (IY+n)

RR A

Esta instrução efetua para a direita uma rotação de todos os bits do acumulador. O bit de reserva toma o lugar do bit 7; ele próprio é substituido pelo bit 0.

Exemplo: MODO DE ENDEREÇAMENTO REGISTRADOR (21 bytes)

Programa BASIC

```
10
20
30 SCREEN 0 : Y = USR0 ( X )
```

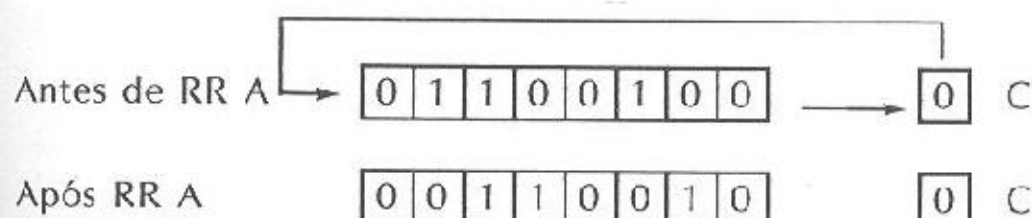
Programa assembler

Linhas	Códigos de máquina	Assembler		
1	06 C8		LD	B,200
2	3E 64		LD	A,100
3	CD A2 00	ROTINA:	CALL	162
4	37		SCF	
5	3F		CCF	
6	CB 1F		RR	A
7	CD A2 00		CALL	162
8	37		SCF	
9	3F		CCF	
10	CB 17		RL	A
11	10 F0		DJNZ	ROTINA (−16)
12	C9		RET	

Linhas 1 e 2: Os registradores B e A são inicializados com os valores 200 e 100. O primeiro terá o papel de contador na rotina ROTINA; o segundo contém, por enquanto, o código ASCII da letra minúscula d.

Linha 3: A rotina de exibição 162 recopia no alto e à esquerda da tela o caractere cujo código está em poder do acumulador. É portanto a letra d que aparece na imagem

Linhas 4 e 5: Obriga-se o bit de reserva, o bit C, a se anular. Desta vez, isso se realiza graças à sucessão das instruções SCF e CCF. A primeira (SCF) força o bit C a tomar o valor 1 e a segunda (CCF) o força a tomar o valor binário oposto (isto é, 0, neste caso). Não existe instrução assembler que sozinha baste para anular o bit de reserva.

Linha 6: O acumulador sofre uma rotação para a sua direita através do indicador de reserva.

A tradução para decimal do novo valor de A é: 50.

Linha 7: O programa retorna de novo a executar o subprograma de exibição; o algarismo 2 (ASCII 50) vai então ser escrito na tela.

Linhas 8 a 10: Nova rotação, mas agora para a esquerda; A retoma então o seu valor de origem, 100.

Linha 11: O processador ficará na rotina ROTINA até que o registrador B se anule. Ver-se-ão neste momento suceder na parte do alto da tela 400 caracteres, a letra d e o algarismo 2 aparecendo alternadamente.

DIFERENTES FORMAS DA ROTAÇÃO PARA A DIREITA (MODO REGISTRADOR)

RR	R	R é um dos registradores de 8 bits.

RR (IX + n)

Uma rotação para a direita do conteúdo de um byte da memória é realizada. O bit 0 toma o lugar do bit de reserva que se encontra, ele próprio, na localização do bit 7.

Exemplo: MODO DE ENDEREÇAMENTO INDIRETO (36 bytes)

Programa BASIC

```
10
20
30 COLOR , 2 : SCREEN 2 : Y = USR0 ( X )
40 GOTO 40
```

Programa assembler

Linhas	Códigos de máquina	Assembler		
1	06 08		LD	B,8
2	DD 21 94 F2		LD	IX,62100
3	DD 36 00 80		LD	(IX+0),128
4	21 00 00	INÍCIO:	LD	HL,0
5	11 B8 0B		LD	DE,3000
6	DD 7E 00	ROTINA:	LD	A,(IX+0)
7	CD CD 07		CALL	1997
8	23		INC	HL
9	1B		DEC	DE
10	7A		LD	A,D
11	B3		OR	E
12	20 F4		JR	NZ,ROTINA (−12)
13	37		SCF	
14	DD CB 00 1E		RR	(IX+0)
15	10 E7		DJNZ	INÍCIO (−25)
16	C9		RET	

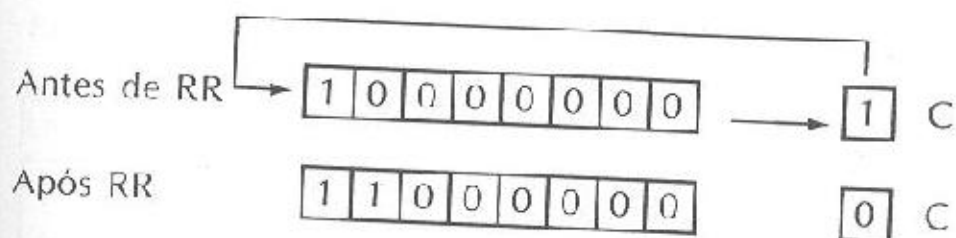
Linhas 1 a 5: B e DE serão úteis como contadores para as rotinas INICIO e ROTINA. Por outro lado, HL aponta sobre o primeiro byte de memória da tela do MSX no modo SCREEN 2. Quanto a IX, ele contém o endereço de um byte no qual está escrito, por enquanto, o valor 128.

Linha 6: Este valor 128 é recopiado no acumulador no início de cada passagem pela rotina ROTINA. Esta forma de agir torna-se necessária pelo fato de que A é utilizado para outra coisa nesta mesma rotina.

Linha 7: Estamos de novo diante do equivalente assembler de VPOKE 0,128. O primeiro segmento do vídeo aparece portanto na tela. Todos os seus pontos são coloridos em verde (cor de fundo) com exceção do seu ponto de esquerda que é azul. Isso se compreende quando se sabe que a conversão binária de 128 dá 10000000.

Linha 8: Tem-se a intenção de iluminar em azul o ponto da esquerda de cada um dos 3 000 primeiros segmentos; HL vai portanto ser incrementado com regularidade...

Linhas 9 a 12: ... enquanto DE é da mesma forma diminuído com regularidade. Quando este par de registradores chegar a 0, os dois registradores que o compõem valerão também 0 e a instrução OR E estabelecerá um OU lógico entre dois valores nulos. O resultado desta operação, nulo também, tornará inoperante a instrução JR NZ da linha 12.

Linha 13: O indicador de reserva é levado a 1 e os bits do byte 62100 são deslocados para a direita através do bit C.

Os 3 000 segmentos sucessivamente atingidos pela chamada do subprograma 1997 vão em seguida se tornar visíveis com seus pontos da esquerda azuis e os outros verdes. Quando o programa terminar, os 8 pontos de cada um destes 3 000 segmentos estarão coloridos de azul.

DIFERENTES FORMAS DA ROTAÇÃO PARA A DIREITA (MODO INDIRETO)

RR (HL)	RR (IX+n)	RR (IY+n)

ADC

Esta instrução é a abreviação de ADd with Carry. Ela é utilizada como a instrução ADD, mas o valor do indicador de reserva C é somado ao resultado da adição. Podem empregar os modos de endereçamento imediato, registrador e indireto.

Exemplo: MODO DE ENDEREÇAMENTO IMEDIATO (19 bytes)

Programa BASIC

```
 10
 20
 30 INPUT "PRIMEIRO NÚMERO " ; N1
 40 POKE 62101 , INT (N1 / 256 )
 50 POKE 62100 , N1 - INT ( N1 / 256) * 256
 60 INPUT " SEGUNDO NÚMERO " ; N2
 70 POKE 62103 , INT ( N2 / 256 )
 80 POKE 62102 , N2 - INT ( N2 / 256 ) * 256
 90 Y = USR0 ( X ) : PRINT " RESPOSTA " ;
100 PRINT 65536*PEEK(62112) + 256*PEEK(62111) + PEEK(62110)
```

Programa assembler

Linhas	Códigos de máquina	Assembler	
1	ED 5B 94 F2	LD	DE,(62100)
2	2A 96 F2	LD	HL,(62102)
3	19	ADD	HL,DE
4	22 9E F2	LD	(62110),HL
5	3E 00	LD	A,0
6	CE 00	ADC	A,0
7	32 A0 F2	LD	(62112),A
8	C9	RET	

Linhas 1 e 4: O registrador de 16 bits DE é carregado com o número N1; é somado a ele o número N2, e o resultado é guardado sob a forma de peso fraco/peso forte, nos bytes 62110 e 62111. O programa poderia parar aí se nos contentássemos de acrescentar dois números tendo uma soma menor que 65536. Suponhamos que isso não aconteça e que propomos ao computador o cálculo 50000 + 20000: ela vai considerar que 70000 se decompõe em 65536 de um lado e em 4464 do outro lado. Este último valor será escrito nos bytes 62110 e 62111 mas conservará a identificação do extravasamento da capacidade de 16 bits forçando a 1 o bit de reserva. Devemos ver como poderemos nos servir desta indicação.

Linhas 5 e 6: Estas duas linhas têm por objetivo escrever no registrador A o algarismo do bit de reserva. Coloca-se o acumulador em 0 e acrescenta-se-lhe então a reserva e o valor 0. No total, A conterá realmente o valor de origem do indicador.

Linha 7: Falta só colocar este resultado no byte 62112, local em que, quando o programa chamar, poderá encontrá-lo.

Em conclusão, se o cálculo da soma ultrapassa 16 bits, o número 65536 é acrescentado ao resultado final pela linha BASIC 100.

DIFERENTES FORMAS DA ADIÇÃO COM RESERVA

<table>
<tr><td>ADC</td><td>A,n8</td><td colspan="4">n8 é um número de 8 bits.</td></tr>
<tr><td>ADC</td><td>A,R</td><td colspan="4">R é um dos registradores de 8 bits.</td></tr>
<tr><td>ADC</td><td>A,(HL)</td><td>ADC</td><td>A,(IX+n)</td><td>ADC</td><td>A,(IY+n)</td></tr>
<tr><td>ADC</td><td>HL,Rd</td><td colspan="4">Rd é um dos registradores duplos BC, DE, HL ou SP.</td></tr>
</table>

LDI

O conteúdo do byte apontado por HL é recopiado no byte apontado por DE. Em seguida estes dois registradores são incrementados.

Exemplo: MODO DE ENDEREÇAMENTO INERENTE (16 bytes)

Programa BASIC

```
10
20
30 FOR I = 62200 TO 62299 : PRINT PEEK ( I ) ;
40 NEXT : Y = USR0 ( X )
50 FOR I = 62200 TO 62299 : PRINT PEEK ( I ) ; : NEXT
```

Programa assembler

Linhas	Códigos de máquina	Assembler		
1	21 94 F2		LD	HL,62100
2	36 01		LD	(HL),1
3	11 F8 F2		LD	DE,62200
4	06 64		LD	B,100
5	ED A0	ROTINA:	LDI	
6	2B		DEC	HL
7	10 FB		DJNZ	ROTINA (−5)
8	C9		RET	

Os bytes do intervalo 62200 — 62299 contém antes da execução de USR0 valores que podem ser iguais a 255 ou a 0. Após a execução do programa assembler, todos estes valores serão iguais a 1. Vejamos porquê.

Linhas 1 e 2: Escreve-se o algarismo 1 no byte 62100 e faz-se de forma que o registrador HL fique durante todo este programa como um apontador deste mesmo byte.

Linha 3: DE contém por enquanto o endereço 62200. É este byte e os seus 99 seguintes que vão ter os seus conteúdos levados a 1.

Linha 5: O valor do byte 62100 (apontado por HL) é recopiado no byte 62200 (apontado por DE). Por enquanto os dois bytes em questão têm o mesmo conteúdo: 1.

Além disso, os registradores HL e DE são automaticamente incrementados. HL passa portanto a 62101 e DE a 62201.

Linha 6: Neste exemplo, deseja-se transferir o conteúdo do byte 62100 para cada um dos bytes que DE apontará sucessivamente. Por isso, HL é diminuído; ele retoma assim o seu valor inicial, 62100.

Linha 7: O programa se desloca de novo para a linha 5 enquanto as 100 passagens pela rotina não forem realizadas. Como se vê sem dificuldade, cada uma destas passagens recopia o valor 1 no byte apontado por DE.

Assim se explica como uma série de 100 algarismos 1 apareça na tela quando se pediu ao BASIC que afixasse os conteúdos dos bytes 62200 a 62299.

Notas.

- A instrução LDD, da mesma forma que LDI, carrega o endereço apontado por DE com o conteúdo do byte apontado por HL, mas efetua em seguida uma diminuição destes dois registradores.
- Por ocasião da execução das instruções LDI e LDD, o registrador BC será sempre diminuído.

CPI

Uma comparação é efetuada entre o byte apontado por HL e o acumulador. Uma instrução de deslocamento deve normalmente acompanhar esta instrução.

Exemplo: MODO DE ENDEREÇAMENTO INERENTE (21 bytes)

Programa BASIC

```
10
20
30 FOR I = 62100 TO 62149 : POKE I,INT (RND(1)*2) : NEXT
40 POKE 62080 , 0 : Y = USR0 ( X )
50 PRINT " O ALGARISMO 0 FOI SORTEADO " ;
70 PRINT PEEK (62080) ; " VEZ(ES) EM 50 "
```

Programa assembler

Linhas	Códigos de máquina	Assembler		
1	DD 21 80 F2		LD	IX,62080
2	21 94 F2		LD	HL,62100
3	06 32		LD	B,50
4	3E 00	INÍCIO:	LD	A,0
5	ED A1		CPI	
6	20 03		JR	NZ,ROTINA (+3)
7	DD 34 00		INC	(IX+0)
8	10 F5	ROTINA:	DJNZ	INÍCIO (−11)
9	C9		RET	

Cinqüenta sorteios que só comportam como resultado os valores 0 e 1 são realizados pela linha BASIC 30. O programa assembler é que vai descontar o número de aparecimentos do algarismo 0. Para isso, colocamos nos bytes 62100 a 62149 os 50 algarismos obtidos pela função RND.

Linhas 1 e 2: O registrador IX aponta para o byte 62080. Este byte será incrementado cada vez que o algarismo 0 aparecer; logo, é aí que o BASIC virá procurar a resposta final. Tivemos o cuidado, naturalmente, de inicializar (linha BASIC 40) o byte 62080 em zero. O registrador HL, por seu lado, contém o endereço do primeiro byte cujo conteúdo se vai analisar.

Linhas 4, 5 e 6: Comparamos, graças a CPI, os conteúdos do acumulador e do byte 62100. Só há duas possibilidades: ou o primeiro número sorteado é o algarismo 1, ou é o algarismo 0. No primeiro caso, a comparação incide sobre dois valores diferentes e JR NZ liga diretamente o computador na linha ROTINA. No segundo caso, a instrução de ligação não tem nenhum efeito, o programa prossegue em seqüência e o conteúdo do byte apontado por IX, portanto o byte 62080, é incrementado.

Linha 8: HL foi automaticamente incrementado pela instrução CPI e contém agora o valor 62101. Na segunda passagem pela rotina INICIO, uma nova comparação será estabelecida entre o conteúdo do acumulador e do byte 62101. Isto conduzirá ao acréscimo de uma unidade ao byte 62080 se (e somente se) o segundo número aleatório for 0.

O programa termina quando os 50 algarismos sorteados forem comparados com 0.

Notas:

- A instrução CPD é equivalente a CPI; ela compara o byte apontado por HL com o acumulador. Mas o registrador HL acha-se em seguida com uma unidade a menos.
- Por ocasião da execução destas duas instruções de comparação, o registrador BC será sempre diminuído.

LDIR

Esta instrução programa a transferência de uma zona da memória para uma outra zona da memória. HL e DE apontam respectivamente para os primeiros bytes de cada uma destas zonas. BC é carregado com o número de bytes a transferir.

Exemplo: MODO DE ENDEREÇAMENTO INERENTE (12 bytes)

Programa BASIC

```
10
20
30 FOR I = 62100 TO 62112 : READ J : POKE I , J : NEXT
40 DATA 0,1,128,0,0,78,79,32,76,73,83,84,0
50 Y = USR0 ( X ) : LIST
```

Programa assembler

Linhas	Códigos de máquina	Assembler
1	21 94 F2	LD HL,62100
2	11 00 80	LD DE,32768
3	01 0D 00	LD BC,13
4	ED B0	LDIR
5	C9	RET

Antes de executar o seu programa no computador, limite-se a digitar estas duas linhas BASIC:

```
5 REM MSX
6 PRINT
```

Em seguida, no modo direto, faça exibir os conteúdos dos bytes 32768 a 32786:

```
FOR I = 32768 TO 32786 : PRINT PEEK ( I ) ; : NEXT
```

Virá:

0 11 128 5 0 143 32 77 83 88	(primeira linha do programa)
0 17 128 6 0 145	(segunda linha do programa
0 0 0	(fim do programa)

Vejamos o sentido destes valores:

11 e 128	dão com a regra peso fraco/peso forte o valor 32779;
5 e 0	dão com essa mesma regra o número 5; é o número da linha BASIC;
143	é o código da palavra reservada REM;
32	é o caractere espaço;
77, 83 e 88	são a tradução ASCII da palavra MSX;
17 e 128	correspondem ao número 32785;
6 e 0	formam o número da segunda linha BASIC;
145	é o código da palavra reservada PRINT.

Os algarismos 0 escritos no início de cada linha funcionam simplesmente como separadores. Observemos que dois zeros suplementares são colocados na memória para indicar o fim do programa.

Falta só compreender o significado dos valores 32779 e 32785. Estes números indicam em que lugar começa a linha BASIC seguinte; faça as contas, você verá que o byte 32779 é o que contém o primeiro elemento (17) da segunda linha do programa.

Voltemos ao assembler: o seu trabalho vai consistir em modificar os primeiros bytes de memória viva para fazer aparecer uma mensagem especial, se por acaso o comando LIST for ativado. Quando os códigos forem executados, eis o que os bytes 32768 a 32780 conterão:

0 1 128 0 0 78 79 32 76 73 83 84 0

Descubra agora o que vai se passar. Entretanto, só mais uma indicaçãozinha: os conteúdos dos bytes 32769 e 32770 correspondem ao valor ... 32769.

Nota: LDDR só é diferente de LDIR pelo fato de que os registradores HL e DE são automaticamente diminuídos.

SRA A

Todos os bits do acumulador são deslocados para a direita e o bit 0 vai para o indicador de reserva. Mas o bit 7 fica inalterado.

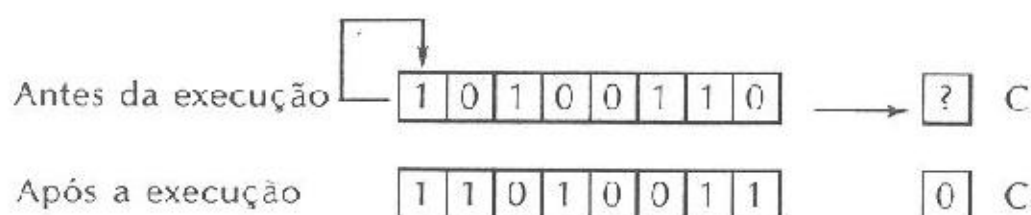
Exemplo: MODO DE ENDEREÇAMENTO REGISTRADOR (9 bytes)

Programa BASIC

```
10
20
30 INPUT " DÊ UM NÚMERO NEGATIVO " ; N
40 POKE 62100 , 256 + N : Y = USR0 ( X )
50 PRINT " EIS O SEU QUOCIENTE POR DOIS " ;
60 PRINT " – " ; 256 – PEEK (62110) : GOTO 30
```

Programa assembler

Linhas	Códigos de máquina	Assembler
1	3A 94 F2	LD A,(62100)
2	CB 2F	SRA A
3	32 9E F2	LD (62110),A
4	C9	RET

É preciso lembrar que os números de 8 bits, cuja escrita binária começa pelo algarismo 1, são considerados pelo computador como negativos. Por exemplo −90 se obtém calculando o complemento da base dois de 90, o que dá 10100110. Examinemos qual será o efeito de SRA neste número se supusermos que ele será escrito no acumulador:

Todos os algarismos foram deslocados para a direita e o último deles passou para o indicador. Quanto ao bit 7, ele valia 1 e no lugar que ele deixou livre, o mesmo algarismo 1 foi escrito. Fazendo o jogo das conversões, obtém-se para A o valor decimal 211. Ora, se procurarmos o complemento da base dois de 45, obteremos justamente 211. Assim, na continuação da execução de SRA, o acumulador contém a tradução binária do valor −45. Fica assim compreendido o papel da nossa nova instrução; ela permite dividir por 2 um número negativo, conservando o seu sinal. Precisamos ver, a nível de BASIC, como podemos fazer chegar ao processador o número a dividir e recuperar em seguida o seu quociente.

N é um número negativo que é necessário transformar pelo modo complemento da base dois. Isso se faz com o POKE da linha 40: com efeito, subtraindo um número de 256, obtém-se o valor decimal de seu complemento da base dois. 255 corresponde por exemplo a −1, 254 a −2 etc.

Retomaremos o mesmo método para traduzir (linha 60) o número negativo, que a máquina tiver calculado, numa forma que nos é habitual.

E por último: não deixe de propor ao computador números ímpares ou números cujo valor absoluto seja superior a 127. Experimente achar cada vez onde se encontra a lógica de uma resposta aparentemente incorreta.

DIFERENTES FORMAS DA INSTRUÇÃO SRA (MODO REGISTRADOR)

SRA	R	R é um dos registradores de 8 bits.

SRA (IX + n)

O conteúdo do byte da memória é submetido a uma rotação para a direita. O bit 7 guarda o seu valor de origem e o bit 0 passa para o indicador de reserva.

Exemplo: MODO DE ENDEREÇAMENTO INDIRETO (27 bytes)

Programa BASIC

```
10
20
30 INPUT " DÊ UM NÚMERO " ; X
40 SCREEN 3 : Y = USR0 ( X )
50 GOTO 50
```

Programa assembler

Linhas	Códigos de máquina		Assembler	
1	CD 8A 2F		CALL	12170
2	06 08		LD	B,8
3	DD 21 94 F2		LD	IX,62100
4	DD 36 00 90		LD	(IX+0),144
5	DD 7E 00	ROTINA:	LD	A,(IX+0)
6	CD CD 07		CALL	1997
7	23		INC	HL
8	DD CB 00 2E		SRA	(IX+0)
9	10 F3		DJNZ	ROTINA (−13)
10	C9		RET	

Linha 1: Utilizamos pela primeira vez um método que transmite diretamente um valor BASIC ao programa assembler. O subprogra-

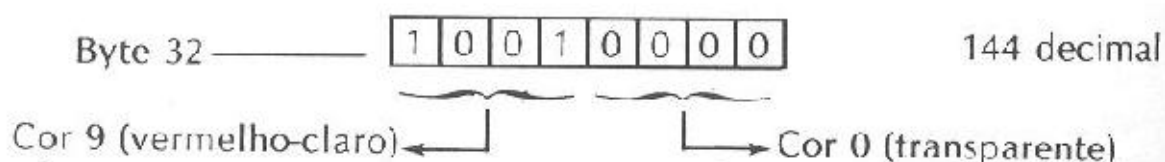
ma 12170 se encarrega de recopiar no registrador HL o valor de 16 bits encontrado na memória na variável X. Suponhamos, por exemplo, que tenhamos respondido "32" à questão "dê um número". O subprograma realiza automaticamente a seguinte instrução:

```
LD HL,32
```

Linhas 3 a 5: O valor 144 é escrito no byte 62100 e uma cópia do conteúdo deste byte é transmitida ao acumulador.

Linha 6: CALL 1997 executa a escrita de um dado contido por A no byte de vídeo apontado por HL. Considerando que X valia 32, a linha 6 é equivalente a:

```
VPOKE 32,144
```

O primeiro retângulo da quinta conluna da tela no modo multicor aparece: é colorido metade de vermelho e metade de azul.

Linhas 7 e 8: O segmento nº 33, colocado precisamente abaixo deste que acabamos de iluminar, é apontado por HL. E o byte 62100, que decididamente já utilizamos muitas vezes neste livro, muda de valor. Imediatamente após a ação de SRA, a configuração dele é a seguinte: 11001000.

Linha 9: Quando o programa assembler tiver terminado, poderemos ver os retângulos ligados aos bytes 32, 33 ... 39 com as seguintes cores:

Byte 32 :	10010000	vermelho e azul
Byte 33 :	11001000	verde e vermelho
Byte 34 :	11100100	cinza e azul
Byte 35 :	11110010	branco e verde
Byte 36 :	11111001	branco e vermelho
Byte 37 :	11111100	branco e verde
Byte 38 :	11111110	branco e cinza
Byte 39 :	11111111	branco e branco

DIFERENTES FORMAS DA INSTRUÇÃO SRA
(MODO INDIRETO)

SRA (HL)	SRA (IX+n)	SRA (IY+n)

CPL

O conteúdo do acumulador é substituído pelo seu complemento lógico. Cada algarismo 1 é transformado num algarismo 0 e reciprocamente.

Exemplo: MODO DE ENDEREÇAMENTO INERENTE (16 bytes)

Programa BASIC

```
10
20
30 SCREEN 2 : OPEN "GRP:" AS1 : PSET ( 0 , 0 )
40 PRINT # 1 , " INVERSÃO DA PRIMEIRA LINHA "
50 FOR I = 1 TO 20 : FOR J = 1 TO 200 : NEXT
60 Y = USR0 ( X ) : NEXT
70 GOTO 70
```

Programa assembler

Linhas	Códigos de máquina	Assembler		
1	06 00		LD	B,0
2	21 00 00		LD	HL,0
3	CD D7 07	ROTINA:	CALL	2007
4	2F		CPL	
5	CD CD 07		CALL	1997
6	23		INC	HL
7	10 F6		DJNZ	ROTINA (−10)
8	C9		RET	

Linhas 1 e 2: Quando o programa tiver terminado a sua execução, 256 passagens pela rotina ROTINA terão sido realizadas e os 256 primeiros bytes da RAM do vídeo no modo SCREEN 2 terão os seus conteúdos modificados.

Linha 3: O processador é lançado na rotina de leitura da memória da tela. Na sua volta, o conteúdo do byte apontado por HL é escrito no acumulador. CALL 2007 é o equivalente assembler da função VPEEK.

Suponhamos que o byte de vídeo nº 0 tenha por conteúdo o número decimal 56, isto é, 00111000 binário. Isso se traduzirá pelo fato de que os seus três pontos centrais serão visíveis em branco no televisor.

Linha 4: Invertem-se todos os algarismos binários deste byte:

Antes de CPL	0	0	1	1	1	0	0	0
Após CPL.	1	1	0	0	0	1	1	1

Linha 5: Recopia-se este novo valor no byte 0. Desta vez, constatamos que são os dois pontos da esquerda e os três pontos da direita do segmento associado que se iluminam em branco. Logo, o primeiro segmento da tela sofreu uma inversão de vídeo.

Linha 7: Repete-se a operação nos 255 segmentos seguintes; cada um deles vai então ser substituído pelo seu complemento lógico. Saindo da rotina ROTINA, cada letra da frase será escrita não mais em branco sobre fundo azul, mas em azul sobre fundo branco. Quando o BASIC retoma o controle do programa, ele deixa escoar um pouco de tempo, depois cede a vez ao assembler. Os nossos 256 segmentos, modificados ainda uma nova vez por complementação, reencontram a sua coloração de origem. O que acabamos de analisar vai reproduzir ainda 9 vezes aos nossos olhos, o tempo exato para que a rotina FOR NEXT seja executada completamente.

NEG

O valor do registrador A é substituído por seu complemento da base dois.

Exemplo: MODO DE ENDEREÇAMENTO INERENTE (9 bytes)

Programa BASIC

```
10 CLEAR 200,62000 : A$ = "CD1F52ED443294F2C9"
20 AD = 62001 : FOR I = 0 TO 8
30 POKE AD + I , VAL("&H"+MID$(A$,2*I+1,2)) : NEXT
40 INPUT " DÊ UM NÚMERO " ; N
50 DEFUSR0 = 62001 : Y = USR0 ( N )
60 PRINT " COMPLEMENTO DA BASE DOIS " ; -N ;
70 PRINT " ESCREVE-SE " ; PEEK(62100) : GOTO 40
```

Programa assembler

Linhas	Códigos de máquina	Assembler
1	CD 1F 52	CALL 21023
2	ED 44	NEG
3	32 94 F2	LD (62100),A
4	C9	RET

Com este programa, encontramo-nos desembaraçados de todos os problemas de escrita dos números negativos pelo modo complemento da base dois. A instrução NEG efetua para nós as duas operações necessárias:

- Complementação lógica,
- Adição de 1 ao resultado obtido.

Linha 1: As rotinas 12170 e 21023 apresentam muitas semelhanças. Mas, enquanto a primeira escreve no registrador HL o valor de uma variável no acumulador. Quando o computador chega à linha 2, o valor N do programa BASIC já está escrito no registrador A.

Linha 2: Procura-se o complemento da base dois de N. Esta linha poderia ser substituída pelas duas instruções assembler seguintes:

```
CPL
ADD A,1
```

Só falta escrever a resposta no byte desejado.
Encontramos poucos programas de máquina tão fáceis de compreender, aproveitemos então o tempo para analisar a forma como os códigos foram carregados pelo BASIC.

Linha BASIC 10: A variável string A$ é formada pela série de códigos CD, 1F, 52..., concatenados uns aos outros.

Linha BASIC 20: Achamos de novo o nosso valor habitual 62001, é neste endereço que será colocado o primeiro código CD. Uma rotina FOR NEXT incidindo sobre a variável I, então, é executada: quando I vale 0, MID$(A$,2*I+1,2) se torna MID$(A$,1,2), isto é, o número hexadecimal CD que POKE colocará no byte 62001. Depois I valerá 1 e, desta vez, POKE inscreverá o código 1F no byte 62002. Isso continuará até que C9 se escreva no byte 62009.

Este método é um pouco menos legível que o utilizado durante todo este livro mas, uma vez que muitos programadores o preferem, era conveniente vê-lo ao menos uma vez.

JP

Abreviação de JumP, esta instrução permite realizar um deslocamento longo para qualquer byte da memória.

Exemplo: MODO DE ENDEREÇAMENTO ABSOLUTO (34 bytes)

Programa BASIC

```
10
20
30 INPUT " DÊ UMA COR " ; C
40 C = C * 16 : SCREEN 2 : Y = USR0 ( C )
50 OPEN " GRP: " AS1 : PRESET ( 0 , 0 )
60 PRINT # 1 , " O REGISTRADOR HL VALIA " ; Y
70 GOTO 70
```

Programa assembler

Linhas 1 e 2: O subprograma 21023 deposita no acumulador o valor da variável C. Admitiremos que repondemos por C = 2 à questão "dê uma cor". A linha BASIC 40 escreveu então em C o valor 32 (00100000 binário) correspondente às cores verde e transparente. O número 32 é em seguida guardado na pilha; vamos encontrá-lo novamente um pouco mais tarde.

Linhas 3 a 6: O registrador HL aponta o primeiro segmento da oitava linha da tela. Então ordena-se que o processador efetue o subprograma LINHA que começa no byte nº 62026 (faça as contas, é mesmo o 26º da nossa série de códigos de máquina). O sub-

Linhas	Códigos de máquina	Assembler		
1	CD 1F 52		CALL	21023
2	F5		PUSH	AF
3	11 08 00		LD	DE,8
4	3E FF		LD	A,255
5	21 00 01		LD	HL,256
6	CD 4A F2		CALL	LINHA (62026)
7	F1		POP	AF
8	21 00 21		LD	HL,8448
9	CD 4A F2		CALL	LINHA (62026)
10	C3 99 2F		JP	12185
11	06 20	LINHA:	LD	B,32
12	CD CD 07	ROTINA:	CALL	1997
13	19		ADD	HL,DE
14	10 FA		DJNZ	ROTINA (−6)
15	C9		RET	

programa escreve o valor 255 no byte 256 e força assim todos os seus bits para 1, em seguida, ele faz o mesmo com o byte 264 (256 + 8 ou HL + DE). Esta ação é repetida 32 vezes ao todo e pode-se portanto estar certo de que a oitava linha do televisor só será constituída de pontos iluminados. O resto do nosso estudo vai nos permitir compreender como a cor destes pontos será escolhida.

Linhas 7 a 8: Reescreve-se o valor 32 no acumulador, carrega-se HL com o endereço do primeiro byte de cor da nossa linha (8448 = 256 + 8192) e volta-se ao nosso subprograma LINHA. Os nossos 32 segmentos vão, portanto, aparecer com a cor verde (todos os seus bits estão no nível 1).

Linha 10: A instrução RET que habitualmente termina os nossos programas foi substituída por JP. Esta realiza um deslocamento incondicional para qualquer byte da memória; neste caso, para nós, trata-se do byte 12185. Este salto reconduz simplesmente ao BASIC mas com um detalhe importante: o computador teve o cuidado de recopiar o valor do registrador HL na variável Y. Por isso se exibe no alto da imagem a frase "O REGISTRADOR HL VALIA 8704".

DIFERENTES FORMAS DA INSTRUÇÃO DE DESVIO JP

JP Z,ENDEREÇO	JP NZ,ENDEREÇO
JP C,ENDEREÇO	JP NC,ENDEREÇO
JP é utilizado neste caso de maneira análoga a JR.	
JP (HL) JP (IX) JP (IY)	
O deslocamento se efetua num endereço contido por HL, IX ou IY.	

RET Z

O retorno ao programa que se chama só se produz quando a instrução precedente for:

- *uma comparação entre dois valores iguais,*
- *uma operação que dá um resultado nulo.*

Exemplo: MODO DE ENDEREÇAMENTO INERENTE (11 bytes)

Programa BASIC

```
10
20
30 SCREEN 0 : Y = USR0 ( X )
```

Programa assembler

Linhas	Códigos de máquina	Assembler		
1	CD 9F 00	INICIO:	CALL	159
2	FE 51		CP	81
3	C8		RET	Z
4	CD A2 00		CALL	162
5	18 F5		JR	INICIO (–11)

Este é um programa que nos autorizará a imprimir sobre o monitor todas as letras que digitarmos.

Linha 1: O microprocessador é enviado à rotina 159. Ficará lá enquanto nenhuma tecla for digitada. Este subprograma faz o ciclo de rotina, portanto sobre ele mesmo, contentando-se de investigar o teclado. Pode-se sair dele apertando uma tecla.

Linha 2: Digitemos por exemplo a letra A; por ora nada aparece na tela, mas ela provoca a saída da rotina 159. Isso não teria nenhuma utilidade se o computador não tivesse a boa idéia de carregar, por conta própria, o código da letra A [ASCII 65] no acumulador antes de encontrar novamente o curso normal do nosso programa. A instrução CP compara por conseguinte os valores 65 e 81.

Linha 3: Como estes valores são diferentes, o comando RET Z é ignorado e é a linha seguinte que se executa.

Linha 4: O subprograma 162 já foi utilizado várias vezes neste livro. Ele realiza a exibição do caractere contido pelo acumulador. Ele realiza a exibição do caractere contido pelo acumulador. É exatamente o que nos convém, não é? Isso explica porque a primeira letra do alfabeto se encontra agora desenhada no alto e à esquerda da tela.

Linha 5: O computador recebe ordem de se deslocar novamente de forma incondicional, à primeira linha. Ele se lança, por isso mesmo, no subprograma de investigação e só sai dele quando é digitada uma tecla. O caractere correspondente aparece então na tela. Isso durará até que se digite a letra Q [ASCII 81]. Assim que o programa se produzir, a linha 2 procederá a uma comparação entre os dois valores iguais e a instrução RET Z (RETORNO se Zero) nos levará de volta ao BASIC.

Naturalmente poderíamos substituir a linha 3 por JRZ, FIM e acrescentar a linha 6 seguinte: FIM:RET. O nosso programa teria feito o mesmo trabalho.

DIFERENTES FORMAS DA INSTRUÇÃO DE RETORNO CONDICIONAL

RET	Z	RET	NZ
RET	C	RET	NC

SET b,A

Esta instrução força a 1 o bit b do acumulador.

Exemplo: MODO DE ENDFREÇAMENTO REGISTRADOR (18 bytes)

Programa BASIC

```
10
20
30 SCREEN 2 : Y = USR0 ( X )
40 GOTO 40
```

Programa assembler

Linhas	Códigos de máquina	Assembler		
1	21 00 20		LD	HL,8192
2	CD D7 07	ROTINA:	CALL	2007
3	CB CF		SET	1,A
4	CD CD 07		CALL	1997
5	23		INC	HL
6	7C		LD	A,H
7	FE 38		CP	56
8	20 F2		JR	NZ,ROTINA(-14)
9	C9		RET	

Linha 1: O registrador HL é carregado com o endereço do primeiro byte da tabela de cores no modo de alta resolução. Esclarecemos que estes bytes contém na inicialização do sistema o valor 4 (você pode obter esta confirmação com a função VPEEK). Isso explica porque a tela nos aparece totalmente colorida de azul-escuro: 4 ou 00000100 binário corresponde com efeito às cores transparente e azul.

Linha 2: O subprograma de leitura da RAM de vídeo retorna, no acumulador, o valor do byte 8192. O registrador A é, por conseguinte, carregado com o valor 4.

Linha 3: A instrução SET 1,A faz com que o segundo bit a partir da direita (bit nº 1) do acumulador tome o valor 1:

4 decimal | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | → SET 1,A ← | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 6 decimal

Linha 4: O computador é lançado na rotina 1997, rotina que é bom lembrar, tem por função modificar a memória do vídeo. O byte apontado por HL é carregado com o valor 6 e o segmento correspondente nos aparece com as cores transparente e vermelha.

Na realidade vemos o nosso segmento inteiramente colorido de vermelho pois ele é constituído só de pontos de fundo.

Linhas 5 a 7: Indica-se a HL que ele deverá cuidar do segundo byte de cor da memória da tela, verifica-se que H não atingiu o valor 56 e remete-se o programa à linha 2. Um novo segmento se tornará então visível no televisor, imitado um pouco mais tarde por cada um dos seus 6 142 seguintes na tabela de cores. Teremos neste momento reconstituído de algum modo a função CLS.

Experimentemos ver em que momento o computador parará de fazer o ciclo da rotina:

- quando HL vale 14335, H vale 55 e L vale 255;
- quando HL vale 14336, H vale 56 e L vale 0.

É claro que basta comparar H com 56 para saber se ainda há bytes a serem modificados pelo programa.

DIFERENTES FORMAS DA INSTRUÇÃO SET

SET b,R	b é o número do bit e R é um dos registradores de 8 bits.
SET b,(HL) SET b,(IX+n) SET b,(IY+n) O bit forçado a 1 é o do byte da memória apontado.	

RES b,A

O bit b do byte contido pelo acumulador é forçado a zero.

Exemplo: MODO DE ENDEREÇAMENTO INDIRETO (25 bytes)

Programa BASIC

```
10
20
30 SCREEN 0 : Y = USR0 ( X )
```

Programa assembler

Linhas	Códigos de máquina	Assembler		
1	21 00 00		LD	HL,0
2	CD D7 07	ROTINA:	CALL	2007
3	CB AF		RES	5,A
4	CB F7		SET	6,A
5	CD CD 07		CALL	1997
6	23		INC	HL
7	7C		LD	A,H
8	FE 03		CP	3
9	20 F0		JR	NZ,ROTINA(–16)
10	7D		LD	A,L
11	FE C0		CP	192
12	20 EB		JR	NZ,ROTINA(–21)
13	C9		RET	

Em modo texto 0, quando a tela está totalmente apagada, cada byte da memória da tela contém o valor 32. Este número é o códi-

go ASCII do caractere branco (espaço). O nosso programa se propõe substituir 32 por 64 e mostrar assim 960 vezes seguidas a letra @.

Linhas 1 e 2: O acumulador é carregado com o conteúdo do byte de vídeo 0, isto é, com o número 32.

Linha 3: O bit nº 5 de A é apagado:

32decimal [0|0|1|0|0|0|0|0] → RES 5,A ← [0|0|0|0|0|0|0|0] 0 decimal

Linha 4: Em compensação, é o bit 6 que se ilumina:

0 decimal [0|0|0|0|0|0|0|0] → SET 6,A ← [0|1|0|0|0|0|0|0] 64 decimal

Linha 5: O subprograma 1997 escreve o valor 64 no byte 0 e a primeira casa de caractere da tela é desenhada: a letra @.

Linha 6: Acrescenta-se uma unidade ao registrador HL e recomeça-se a executar as instruções da rotina ROTINA. Uma segunda, depois uma terceira, depois uma 960ª letra serão então exibidas. O trabalho efetuado por este programa é praticamente fácil de analisar; o único problema a observar com mais cuidado é talvez a forma como se deve indicar ao computador quando ele deveria parar a rotina.

Linha 9: Envia-se de novo o processador à linha ROTINA se o registrador H não for igual a 3. Como 3*256 = 768, pode-se deduzir que o programa não ultrapassa a linha 9 enquanto HL não atingir 768.

Linhas 10 a 12: Em seguida, a partir deste momento, vigia-se o registrador L. Assim que ele chegar a 192, sai-se da rotina. Com efeito:

3 (registrador H) * 256 + 192 (registrador L) = 960 (registrador HL)

DIFERENTES FORMAS DA INSTRUÇÃO RES

RES	b,R	b é o número do bit, R um dos registradores de 8 bits.			
RES	b,(HL)	RES	b,(IX+n)	RES	b,(IY+n)
O bit b do byte de memória apontado é reduzido a 0.					

EX DE,HL

Esta instrução programa a troca dos registradores DE e HL.

Exemplo: MODO DE ENDEREÇAMENTO REGISTRADOR (22 bytes)

Programa BASIC

```
10
20
30 SCREEN 0 : WIDTH ( 40 ) : Y = USR0 ( X )
```

Programa assembler

Linhas	Códigos de máquina	Assembler		
1	11 20 00		LD	DE,32
2	21 FF 00		LD	HL,255
3	0E 14		LD	C,20
4	06 28	INICIO:	LD	B,40
5	7D	ROTINA:	LD	A,L
6	CD A2 00		CALL	162
7	EB		EX	DE,HL
8	10 F9		DJNZ	ROTINA (−7)
9	EB		EX	DE,HL
10	0D		DEC	C
11	20 F3		JR	NZ,INICIO (−13)
12	C9		RET	

Linhas 1 e 2: Escrever 32 e 255 (valores de 8 bits) no pares de registradores DE e HL vem a ser o mesmo que carregar os registradores E e L com estes valores e colocar o número 0 nos registradores D e H.

Linhas 3 e 4: B contém o número máximo de caracteres que se

podem escrever numa linha de tela e C o número de linhas nas quais a exibição vai ser realizada.

Linhas 5 e 6: O conteúdo do acumulador é levado a 255 e o subprograma 162 é chamado. O seu papel é mostrar o caractere cujo código é escrito no acumulador. Um quadradinho branco — CHR$(255) — se desenha portanto no alto e à esquerda da imagem. O cursor, sem que tenhamos que fazer nada, é deslocado de uma posição para a direita.

Linha 7: Nossa nova instrução provoca a troca dos conteúdos de DE e de HL. DE toma por conseguinte o valor 255 e HL o valor 32. Isto quer dizer que E e L tornaram-se iguais, respectivamente, a 255 e 32.

Linha 8: Diminuindo o registrador B, verifica-se que não chegou ao fim da linha e, como não é o caso, liga-se de novo o programa na linha 5. A nova chamada da rotina de exibição vai fazer aparecer um espaço, isto é, um caractere no qual nada é escrito. É um caractere vazio, de cor azul, portanto, que é agora colocado à direita do quadradinho branco! Como o que acabamos de ver vai se reproduzir até o fim da linha, pode-se deduzir que o objetivo do nosso programa é desenhar um tabuleiro de xadrez no televisor.

Linha 9: O último caractere da primeira linha é azul e se deixarmos o computador continuar a sua marcha, ele mostrará um caractere branco no início da segunda linha, e o efeito do xadrez não será obtido. É por isso que se teve que decidir por uma troca suplementar dos registradores E e L.

Linhas 10 e 11: O programa encontra de novo o bloco de instruções INICIO até que o registrador C, à força de diminuições sucessivas, tome o valor 0. Neste ponto, veremos no televisor a totalidade do nosso tabuleiro de xadrez: ele terá 40 caracteres de largura por 20 de altura.

DIFERENTES FORMAS DA INSTRUÇÃO DE TROCA

EX	DE,HL				
EX	(SP),HL	EX	(SP),IX	EX	(SP),IY
A troca é realizada entre a memória apontada por SP e um dos registradores HL, IX ou IY.					

CONCLUSÃO

Este livro constituiu uma introdução à programação em linguagem de máquina do computador MSX. Estudamos os seus mais importantes aspectos e realizamos uma série de exercícios que lhe mostraram, e era este o nosso desejo, que o assembler podia ser assimilado sem dificuldade por um leitor munido apenas de boa-vontade. Quanto a nós, estamos persuadidos de que é infinitamente mais demorado adquirir a lógica da programação BASIC do que a do assembler.

Agora você está em condições de criar os seus próprios programas e incluir nas linhas BASIC efeitos especiais que somente a impressionante rapidez do assembler permite. Se houver oportunidade, não deixe de procurar a que correspondem os códigos que outros programadores obtiveram, fazendo assim o trabalho inverso do que foi efetuado até aqui. Esta operação lhe permite reconstituir o programa e eventualmente modificá-lo para que ele se adapte com muita precisão ao seu caso.

Naturalmente, nada o impede de ultrapassar uma nova etapa orientando-se para obras mais técnicas do que esta. Nelas você encontrará programas aplicáveis à gestão dos periféricos bem como explicações referentes a algumas instruções sobre as quais voluntariamente passamos em silêncio, julgando que, num primeiro momento pelo menos, o seu interesse era negligenciável.

Talvez você sinta agora a necessidade de comprar o cassete Zen contendo o programa editor/assembler do MSX. Ele realizará para você, sem risco de erro e muito rapidamente, a tradução para linguagem de máquina dos programas escritos em assembler. Trata-se de auxiliar indispensável a todos os que descobriram, com entusiasmo, que podemos nos dirigir diretamente a um microprocessador.

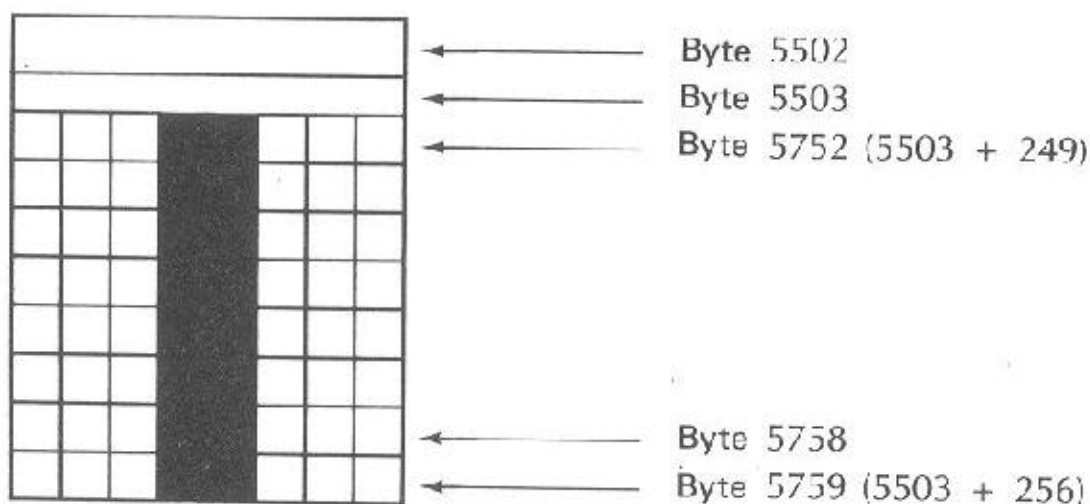
Este livro se encerra com três programas um pouco mais complicados do que os outros. Você os abordará sem receio, agora que lhe foi aberto o estreito, mas tão nobre caminho do assembler.

APÊNDICE A
DESLOCAMENTO DE UM MÓVEL NA TELA

Programa BASIC (58 bytes)

```
 10
 20
 30 COLOR 4,1,4 : SCREEN 2,2 : OPEN "GRP:" AS1
 40 A$ = "" : FOR I = 1 TO 24 : READ J : A$ = A$ + CHRS(J) : NEXT
 50 DATA 0,128,67,52,15,16,16,0
 60 DATA 0,0,0,0,0,0,0,0
 70 DATA 0,1,194,44,240,8,8,0
 80 SPRITE$ (0) = A$
 90 PSET ( 120,191 ) : DRAW "E2U5R1U8R1D8R1D5F2"
100 FOR I = 1 TO INT (RND(1)×3000) : NEXT
110 FOR X = 0 TO 200 STEP 2 : PUTSPRITE 0,(X,0),8,0
120 IF STRIG(0) = 0 THEN NEXT : GOTO 90
130 Y = USR0 ( X ) : COLOR 3 : PSET ( 120 , 100 )
140 IF X > 108 AND X < 125 THEN PRINT # 1 , " GANHOU "
    ELSE PRINT # 1 , " PERDEU "
150 FOR I = 1 TO 1000 : NEXT
160 LINE ( 120 , 100 ) - ( 160 , 127 ) , 1 , BF
170 COLOR 4 : GOTO 90
```

Programa assembler

Este programa desloca um míssil de baixo para cima da tela ao encontro de um disco em movimento. A forma do míssil (ver o desenho) foi simplificada ao máximo para que as explicações pudessem ser compreendidas sem muita dificuldade.

Linhas	Códigos de máquina	Assembler		
1	21 80 15		LD	HL,5504
2	0E 15		LD	C,21
3	06 08	INICIO:	LD	B,8
4	2B	CASA:	DEC	HL
5	E5		PUSH	HL
6	3E 18		LD	A,24
7	CD CD 07		CALL	1997
8	11 00 01		LD	DE,256
9	19		ADD	HL,DE
10	3E 00		LD	A,0
11	CD CD 07		CALL	1997
12	CD 65 F2		CALL	TEMPO (62053)
13	E1		POP	HL
14	10 EA		DJNZ	CASA (−22)
15	0D		DEC	C
16	28 09		JR	Z,FIM (+9)
17	37		SCF	
18	3F		CCF	
19	11 F8 00		LD	DE,248
20	ED 52		SBC	HL,DE
21	18 DC		JR	INICIO (−36)
22	06 08	FIM	LD	B,8
23	3E 00		LD	A,0
24	CD CD 07	APAGA:	CALL	1997
25	23		INC	HL
26	10 FA		DJNZ	APAGA (−6)
27	C9		RET	
28	3E 00	TEMPO:	LD	A,0
29	3D	ESPERA:	DEC	A
30	20 FD		JR	NZ,ESPERA (−3)
31	C9		RET	

Linhas 4 a 7: O valor 24 é escrito no byte 5503 e os dois pontos centrais do segmento correspondente se tornam então visiveis.

Linhas 8 a 11: Acrescentando 256 a HL, faz-se com que este registrador aponte no elemento mais baixo do míssil. A escrita do algarismo 0 no byte 5759 tornará invisível o segmento que lhe é as-

sociado. Logo, faz-se avançar por enquanto o nosso móvel de um pixel para cima.

Linha 12: Deixa-se passar um pouco de tempo lançando o processador na execução de uma rotina vazia.

Linhas 13 e 14: Dá-se novamente a HL o valor 5503 e retorna-se à rotina CASA. Como HL é diminuido, desta vez os dois pontos centrais do segmento 5502 vão ser coloridos, ao passo que os dois pontos iluminados do segmento 5758 (5502 + 256) vão desaparecer da imagem. O pequeno foguete terá então avançado uma posição para cima.

Linhas 15 a 21: Quando o programa chegar a estas linhas, o míssil já terá sofrido oito translações elementares para cima. Ele terá sido então deslocado numa altura de um caractere. Só lhe faltará continuar o seu caminho ascendente até atingir o alto do televisor (o contador C passará então a 0).

Linhas 22 a 27: Antes de deixar que o BASIC retome a direção do programa, deve-se apagar da tela o missil que chegou ao fim da corrida. Para isso, basta escrever o valor 0 em cada um dos oito bytes que constituem o corpo do míssil.

Nota: A instrução SBC HL,DE da linha 20 efetua uma subtração entre os registradores HL e DE (resultado em HL). Mas, além disso, o valor do bit de reserva, C, é dimuído do resultado. É por esta razão, que tomamos o cuidado de forçar esta indicação a zero com o par de comandos SCF — CCF. Observe que infelizmente a instrução SUB HL,DE é desconhecida para o microprocessador Z80.

APÊNDICE B
TRI
EM MEMÓRIA CENTRAL

Programa BASIC (27 bytes)

```
10
20
30 FOR I = 62100 TO 62149 : X = INT (RND(1)*250)
40 PRINT X ; : POKE I , X : NEXT : PRINT
50 Y = USR0 ( X ) : FOR I = 62100 TO 62149
60 PRINT PEEK (I) ; : NEXT
```

Programa assembler

Linhas	Códigos de máquina	Assembler		
1	21 94 F2		LD	HL,62100
2	06 31		LD	B,49
3	C5	INICIO:	PUSH	BC
4	54		LD	D,H
5	5D		LD	E,L
6	13		INC	DE
7	1A	PASSE:	LD	A,(DE)
8	BE		CP	(HL)
9	30 06		JR	NC,NADA
10	1A		LD	A,(DE)
11	F5		PUSH	AF
12	7E		LD	A,(HL)
13	12		LD	(DE),A
14	F1		POP	AF
15	77		LD	(HL),A
16	13	NADA:	INC	DE
17	10 F3		DJNZ	PASSE
18	23		INC	HL
19	C1		POP	BC
20	10 EB		DJNZ	INICIO
21	C9		RET	

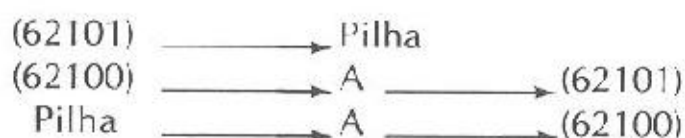
O programa BASIC coloca nos bytes cujos endereços se escalonam entre 62100 e 62149, cinqüenta valores sorteados. O assembler tem a tarefa de colocar ordem nesta lista e retornar os 50 bytes organizados em ordem crescente.

Linhas 1 e 2: O registrador HL é carregado com o endereço do primeiro elemento da lista e B com um valor que é igual ao número de passagens que o programa vai efetuar na rotina INICIO.

Linhas 4 a 6: Escreve-se em DE o número 62101 ...

Linhas 7 a 9: ... e comparam-se os conteúdos dos bytes 62101 e 62100.

Linhas 10 a 15: Se o segundo elemento da seqüência é superior ao primeiro, nenhuma ação é feita e o programa prossegue o seu desenvolvimento na linha NADA. Caso contrário, procede-se a uma permutação entre os conteúdos dos bytes 62100 e 62101. Realiza-se esta troca em três fases.

(62101) ⟶ Pilha
(62100) ⟶ A ⟶ (62101)
Pilha ⟶ A ⟶ (62100)

Linhas 16 e 17: Agora estamos certos de que o primeiro número é inferior ou igual ao segundo.

A incrementação de DE faz com que o registrador aponte desde já sobre o terceiro elemento da série. Quando o computador tiver passado pela segunda vez na rotina PASSE, estaremos aptos a afirmar que o primeiro elemento também é inferior ao terceiro. Na 49ª passagem, teremos a certeza de que o elemento nº 1 é inferior a todos os outros.

Linhas 18 a 20: Faz-se apontar HL para o segundo número da lista e relança-se o programa na linha 3. Compara-se sucessivamente o segundo elemento com os seus 48 subseqüentes, efetuando uma troca quando estes últimos lhe são inferiores. No fim desta etapa, os dois primeiros números são menores do que todos os outros e também são organizados em ordem crescente.

Falta então comparar o terceiro, o quarto ... com os números que os seguem na lista.

APÊNDICE C
MOVIMENTAÇÃO DE UM SPRITE

Programa BASIC (33 bytes)

```
10
20
30 SCREEN 2 , 1 : A $ = ""
40 FOR I = 1 TO 8 : READ J : A$ = A$ + CHR$ ( J ) : NEXT
50 DATA 24,153,66,60,24,24,36,36
60 VPOKE 6913 , 100 : VPOKE 6915 , 3
70 SPRITE$ ( 0 ) = A$ : Y = USR0 ( X )
```

Programa assembler

Linhas	Códigos de máquina	Assembler		
1	21 00 1B		LD	HL,6912
2	06 0A		LD	B,10
3	3E F0	PARE:	LD	A,240
4	CD CD 07	ROTINA:	CALL	1997
5	CD 46 F2		CALL	TEMPO (62022)
6	3C		INC	A
7	FE C0		CP	192
8	20 F5		JR	NZ,ROTINA (−11)
9	10 F1		DJNZ	PARE (−15)
10	C9		RET	
11	F5	TEMPO:	PUSH	AF
12	11 00 04		LD	DE,1024
13	1B	ESPERA:	DEC	DE
14	7A		LD	A,D
15	FE 00		CP	0
16	20 FA		JR	NZ,ESPERA (−6)
17	F1		POP	AF
18	C9		RET	

No modo SCREEN 2, o computador possui uma série de bytes da RAM do video reservada aos sprites. Descontam-se 4 bytes para o sprite 0, 4 para o sprite 1 etc. O primeiro byte da tabela dos sprites tem por endereço 6912. Vejamos a que corresponde este byte e os três seguintes:

- byte 6912: ordenada da figura;
- byte 6913: abscissa da figura;
- byte 6914: nenhum interesse para o nosso exemplo;
- byte 6915: cor da figura.

A linha BASIC 60 escreve no byte 6915 o valor 3 (o sprite 0 será de cor verde) e no byte 6913 o número 100 (o sprite 0 terá uma abscissa igual a 100).

Linhas 1 e 2: HL conservará o mesmo valor durante todo o programa: ele apontará sobre o byte que define a ordenada da figura 0. Quanto a B, a sua presença só serve para nos dar oportunidade de ver uma dezena de vezes a mesma animação.

Linha 3: 240 é a tradução para o processador do número negativo −16 (complemento da base 2).

Linha 4: O subprograma 1997 de escrita na memória da tela carrega o byte 6912 com o número 240. A figura se coloca então no ponto de coordenadas (100, −16). Naturalmente, nenhum dos seus pontos é visível por enquanto, mas esperemos a continuação.

Linhas 7 e 8: A comparação entre 241 e 192 incidindo sobre dois números diferentes, o programa se desloca de novo à linha ROTINA. Mas como entrementes o acumulador foi incrementado, a figura 0 é colocada de forma que sua borda superior esquerda tenha por coordenadas (100, −15). O seu limite inferior começa portanto a aparecer na imagem. Na terceira passagem pela rotina, veremos um pouco mais, depois ainda um pouco mais etc.

Linha 9: Quando o conteúdo do acumulador atingir o valor 192, a figura terá desaparecido completamente na parte inferior do televisor. Bastará recolocar em A o número 240 para fazê-la reaparecer na parte superior da tela.

APÊNDICE D
JOGO DE INSTRUÇÕES DO Z80

Código Objeto	Instrução	
8E	ADC	A,(HL)
DD8E05	ADC	A,(IX+d)
FD8E05	ADC	A,(IY+d)
8F	ADC	A,A
88	ADC	A,B
89	ADC	A,C
8A	ADC	A,D
8B	ADC	A,E
8C	ADC	A,H
8D	ADC	A,L
CE20	ADC	A,n
ED4A	ADC	HL,BC
ED5A	ADC	HL,DE
ED6A	ADC	HL,HL
ED7A	ADC	HL,SP
86	ADD	A,(HL)
DD8605	ADD	A,(IX+d)
FD8605	ADD	A,(IY+d)
87	ADD	A,A
80	ADD	A,B
81	ADD	A,C
82	ADD	A,D
83	ADD	A,E
84	ADD	A,H
85	ADD	A,L
C620	ADD	A,n
09	ADC	HL,BC
19	ADC	HL,DE
29	ADC	HL,HL
39	ADC	HL,SP
DD09	ADD	IX,BC
DD19	ADD	IX,DE
DD29	ADD	IX,IX
DD39	ADD	IX,SP
FD09	ADD	IY,BC
FD19	ADD	IY,DE
FD29	ADD	IY,IY
FD39	ADD	IY,SP
A6	AND	(HL)
DDA605	AND	(IX+d)
FDA605	AND	(IY+d)
A7	AND	A
A0	AND	B
A1	AND	C
A2	AND	D
A3	AND	E
A4	AND	H
A5	AND	L

Código Objeto	Instrução	
E620	AND	n
CB46	BIT	0,(HL)
DDCB0546	BIT	0,(IX+d)
FDCB0546	BIT	0,(IY+d)
CB47	BIT	0,A
CB40	BIT	0,B
CB41	BIT	0,C
CB42	BIT	0,D
CB43	BIT	0,E
CB44	BIT	0,H
CB45	BIT	0,L
CB4E	BIT	1,(HL)
DDCB054E	BIT	1,(IX+d)
FDCB054E	BIT	1,(IY+d)
CB4F	BIT	1,A
CB48	BIT	1,B
CB49	BIT	1,C
CB4A	BIT	1,D
CB4B	BIT	1,E
CB4C	BIT	1,H
CB4D	BIT	1,L
CB56	BIT	2,(HL)
DDCB0556	BIT	2,(IX+d)
FDCB0556	BIT	2,(IY+d)
CB57	BIT	2,A
CB50	BIT	2,B
CB51	BIT	2,C
CB52	BIT	2,D
CB53	BIT	2,E
CB54	BIT	2,H
CB55	BIT	2,L
CB5E	BIT	3,(HL)
DDCB055E	BIT	3,(IX+d)
FDCB055E	BIT	3,(IY+d)
CB5F	BIT	3,A
CB58	BIT	3,B
CB59	BIT	3,C
CB5A	BIT	3,D
CB5B	BIT	3,E
CB5C	BIT	3,H
CB5D	BIT	3,L
CB66	BIT	4,(HL)
DDCB0566	BIT	4,(IX+d)
FDCB0566	BIT	4,(IY+d)
CB67	BIT	4,A
CB60	BIT	4,B
CB61	BIT	4,C
CB62	BIT	4,D

Código Objeto	Instrução	
CB63	BIT	4,E
CB64	BIT	4,H
CB65	BIT	4,L
CB6E	BIT	5,(HL)
DDCB056E	BIT	5,(IX+d)
FDCB056E	BIT	5,(IY+d)
CB6F	BIT	5,A
CB68	BIT	5,B
CB69	BIT	5,C
CB6A	BIT	5,D
CB6B	BIT	5,E
CB6C	BIT	5,H
CB6D	BIT	5,L
CB76	BIT	6,(HL)
DDCB0576	BIT	6,(IX+d)
FDCB0576	BIT	6,(IY+d)
CB77	BIT	6,A
CB70	BIT	6,B
CB71	BIT	6,C
CB72	BIT	6,D
CB73	BIT	6,E
CB74	BIT	6,H
CB75	BIT	6,L
CB7E	BIT	7,(HL)
DDCB057E	BIT	7,(IX+d)
FDCB057E	BIT	7,(IY+d)
CB7F	BIT	7,A
CB78	BIT	7,B
CB79	BIT	7,C
CB7A	BIT	7,D
CB7B	BIT	7,E
CB7C	BIT	7,H
CB7D	BIT	7,L
DC8405	CALL	C,nn
FC8405	CALL	M,nn
D48405	CALL	NC,nn
C48405	CALL	NZ,nn
F48405	CALL	P,nn
EC8405	CALL	PE,nn
E48405	CALL	PO,nn
CC8405	CALL	Z,nn
CD8405	CALL	nn
3F	CCF	
BE	CP	(HL)
DDBE05	CP	(IX+d)
FDBE05	CP	(IY+d)
BF	CP	A
B8	CP	B
B9	CP	C
BA	CP	D
BB	CP	E
BC	CP	H
BD	CP	L
FE20	CP	n
EDA9	CPD	
EDB9	CPDR	
EDB1	CPIR	
EDA1	CPI	
2F	CPL	
27	DAA	
35	DEC	(HL)
DD3505	DEC	(IX+d)
FD3505	DEC	(IY+d)
3D	DEC	A
05	DEC	B
0B	DEC	BC
0D	DEC	C
15	DEC	D
1B	DEC	DE
1D	DEC	E
25	DEC	H
2B	DEC	HL
DD2B	DEC	IX
FD2B	DEC	IY
2D	DEC	L
3B	DEC	SP
F3	DI	
102E	DJNZ	e
FB	EI	
E3	EX	(SP),HL
DDE3	EX	(SP),IX
FDE3	EX	(SP),IY
08	EX	AF,AF'
EB	EX	DE,HL
D9	EXX	
76	HALT	
ED46	IM	0
ED56	IM	1
ED5E	IM	2
ED78	IN	A,(C)
ED40	IN	B,(C)
ED48	IN	C,(C)
ED50	IN	D,(C)
ED58	IN	E,(C)
ED60	IN	H,(C)
ED68	IN	L,(C)
34	INC	(HL)
DD3405	INC	(IX+d)
FD3405	INC	(IY+d)
3C	INC	A
04	INC	B
03	INC	BC
0C	INC	C
14	INC	D
13	INC	DE
1C	INC	E
24	INC	H
23	INC	HL
DD23	INC	IX
FD23	INC	IY
2C	INC	L
33	INC	SP
DB20	IN	A,(n)

Código Objeto	Instrução	
EDAA	IND	
EDBA	INDR	
EDA2	INI	
EDB2	INIR	
C38405	JP	nn
E9	JP	(HL)
DDE9	JP	(IX)
FDE9	JP	(IY)
DA8405	JP	C,nn
FA8405	JP	M,nn
D28405	JP	NC,nn
C28405	JP	NZ,nn
F28405	JP	P,nn
EA8405	JP	PE,nn
E28405	JP	PO,nn
CA8405	JP	Z,nn
382E	JR	C,e
302E	JR	NC,e
202E	JR	NZ,e
282E	JR	Z,e
182E	JR	e
02	LD	(BC),A
12	LD	(DE),A
77	LD	(HL),A
70	LD	(HL),B
71	LD	(HL),C
72	LD	(HL),D
73	LD	(HL),E
74	LD	(HL),H
75	LD	(HL),L
3620	LD	(HL),n
DD7705	LD	(IX+d),A
DD7005	LD	(IX+d),B
DD7105	LD	(IX+d),C
DD7205	LD	(IX+d),D
DD7305	LD	(IX+d),E
DD7405	LD	(IX-d),H
DD7505	LD	(IX+d),L
DD360520	LD	(IX+d),n
FD7705	LD	(IY+d),A
FD7005	LD	(IY+d),B
FD7105	LD	(IY+d),C
FD7205	LD	(IY+d),D
FD7305	LD	(IY+d),E
FD7405	LD	(IY+d),H
FD7505	LD	(IY+d),L
FD360520	LD	(IY-d),n
328405	LD	(nn),a
ED438405	LD	(nn),BC
ED538405	LD	(nn),DE
228405	LD	(nn),HL
DD228405	LD	(nn),IX
FD228405	LD	(nn),IY
ED738405	LD	(nn),SP
0A	LD	A,(BC)
1A	LD	A,(DE)
7E	LD	A,(HL)

Código Objeto	Instrução	
DD7E05	LD	A,(IX+d)
FD7E05	LD	A,(IY+d)
3A8405	LD	A,(nn)
7F	LD	A,A
78	LD	A,B
79	LD	A,C
7A	LD	A,D
7B	LD	A,E
7C	LD	A,H
ED57	LD	A,I
7D	LD	A,L
3E20	LD	A,n
ED5F	LD	A,R
46	LD	B,(HL)
DD4605	LD	B,(IX+d)
FD4605	LD	B,(IY+d)
47	LD	B,A
40	LD	B,B
41	LD	B,C
42	LD	B,D
43	LD	B,E
44	LD	B,H
45	LD	B,L
0620	LD	B,n
ED4B8405	LD	BC,(nn)
018405	LD	BC,nn
4E	LD	C,(HL)
DD4E05	LD	C,(IX+d)
FD4E05	LD	C,(IY+d)
4F	LD	C,A
48	LD	C,B
49	LD	C,C
4A	LD	C,D
4B	LD	C,E
4C	LD	C,H
4D	LD	C,L
0E20	LD	C,n
56	LD	D,(HL)
DD5605	LD	D,(IX-d)
FD5605	LD	D,(IY+d)
57	LD	D,A
50	LD	D,B
51	LD	D,C
52	LD	D,D
53	LD	D,E
54	LD	D,H
55	LD	D,L
1620	LD	D,n
ED5B8405	LD	DE,(nn)
118405	LD	DE,nn
5E	LD	E,(HL)
DD5E05	LD	E,(IX+d)
FD5E05	LD	E,(IY+d)
5F	LD	E,A
58	LD	E,B
59	LD	E,C
5A	LD	E,D

Código Objeto	Instrução	
5B	LD	E,E
5C	LD	E,H
5D	LD	E,L
1E20	LD	E,n
66	LD	H,(HL)
DD6605	LD	H,(IX+d)
FD6605	LD	H,(IY+d)
67	LD	H,A
60	LD	H,B
61	LD	H,C
62	LD	H,D
63	LD	H,E
64	LD	H,H
65	LD	H,L
2620	LD	H,n
2A8405	LD	HL,(nn)
218405	LD	HL,nn
ED47	LD	I,A
DD2A8405	LD	IX,(nn)
DD218405	LD	IX,nn
FD2A8405	LD	IY,(nn)
FD218405	LD	IY,nn
6E	LD	L,(HL)
DD6E05	LD	L,(IX+d)
FD6E05	LD	L,(IY+d)
6F	LD	L,A
68	LD	L,B
69	LD	L,C
6A	LD	L,D
6B	LD	L,E
6C	LD	L,H
6D	LD	L,L
2E20	LD	L,n
ED4F	LD	R,A
ED7B8405	LD	SP,(nn)
F9	LD	SP,HL
DDF9	LD	SP,IX
FDF9	LD	SP,IY
318405	LD	SP,nn
EDA8	LDD	
EDB8	LDDR	
EDA0	LDI	
EDB0	LDIR	
ED44	NEG	
00	NOP	
B6	OR	(HL)
DDB605	OR	(IX+d)
FDB605	OR	(IY+d)
B7	OR	A
B0	OR	B
B1	OR	C
B2	OR	D
B3	OR	E
B4	OR	H
B5	OR	L
F620	OR	n
EDBB	OTDR	

Código Objeto	Instrução	
EDB3	OTIR	
ED79	OUT	(C),A
ED41	OUT	(C),B
ED49	OUT	(C),C
ED51	OUT	(C),D
ED59	OUT	(C),E
ED61	OUT	(C),H
ED69	OUT	(C),L
D320	OUT	(n),A
EDAB	OUTD	
EDA3	OUTI	
F1	POP	AF
C1	POP	BC
D1	POP	DE
E1	POP	HL
DDE1	POP	IX
FDE1	POP	IY
F5	PUSH	AF
C5	PUSH	BC
D5	PUSH	DE
E5	PUSH	HL
DDE5	PUSH	IX
FDE5	PUSH	IY
CB86	RES	0,(HL)
DDCB0586	RES	0,(IX+d)
FDCB0586	RES	0,(IY+d)
CB87	RES	0,A
CB80	RES	0,B
CB81	RES	0,C
CB82	RES	0,D
CB83	RES	0,E
CB84	RES	0,H
CB85	RES	0,L
CB8E	RES	1,(HL)
DDCB058E	RES	1,(IX+d)
FDCB058E	RES	1,(IY+d)
CB8F	RES	1,A
CB88	RES	1,B
CB89	RES	1,C
CB8A	RES	1,D
CB8B	RES	1,E
CB8C	RES	1,H
CB8D	RES	1,L
CB96	RES	2,(HL)
DDCB0596	RES	2,(IX+d)
FDCB0596	RES	2,(IY+d)
CB97	RES	2,A
CB90	RES	2,B
CB91	RES	2,C
CB92	RES	2,D
CB93	RES	2,E
CB94	RES	2,H
CB95	RES	2,L
CB9E	RES	3,(HL)
DDCB059E	RES	3,(IX+d)
FDCB059E	RES	3,(IY+d)

Código Objeto	Instrução	
CB9F	RES	3,A
CB98	RES	3,B
CB99	RES	3,C
CB9A	RES	3,D
CB9B	RES	3,E
CB9C	RES	3,H
CB9D	RES	3,L
CBA6	RES	4,(HL)
DDCB05A6	RES	4,(IX+d)
FDCB05A6	RES	4,(IY+d)
CBA7	RES	4,A
CBA0	RES	4,B
CBA1	RES	4,C
CBA2	RES	4,D
CBA3	RES	4,E
CBA4	RES	4,H
CBA5	RES	4,L
CBAE	RES	5,(HL)
DDCB05AE	RES	5,(IX+d)
FDCB05AE	RES	5,(IY+d)
CBAF	RES	5,A
CBA8	RES	5,B
CBA9	RES	5,C
CBAA	RES	5,D
CBAB	RES	5,E
CBAC	RES	5,H
CBAD	RES	5,L
CBB6	RES	6,(HL)
DDCB05B6	RES	6,(IX+d)
FDCB05B6	RES	6,(IY+d)
CBB7	RES	6,A
CBB0	RES	6,B
CBB1	RES	6,C
CBB2	RES	6,D
CBB3	RES	6,E
CBB4	RES	6,H
CBB5	RES	6,L
CBBE	RES	7,(HL)
DDCB05BE	RES	7,(IX+d)
FDCB05BE	RES	7,(IY+d)
CBBF	RES	7,A
CBB8	RES	7,B
CBB9	RES	7,C
CBBA	RES	7,D
CBBB	RES	7,E
CBBC	RES	7,H
CBBD	RES	7,L
C9	RET	
D8	RET	C
F8	RET	M
D0	RET	NC
C0	RET	NZ
F0	RET	P
E8	RET	PE
E0	RET	PO
C8	RET	Z

Código Objeto	Instrução	
ED4D	RETI	
ED45	RETN	
CB16	RL	(HL)
DDCB0516	RL	(IX+d)
FDCB0516	RL	(IY+d)
CB17	RL	A
CB10	RL	B
CB11	RL	C
CB12	RL	D
CB13	RL	E
CB14	RL	H
CB15	RL	L
17	RLA	
CB06	RLC	(HL)
DDCB0506	RLC	(IX+d)
FDCB0506	RLC	(IY+d)
CB07	RLC	A
CB00	RLC	B
CB01	RLC	C
CB02	RLC	D
CB03	RLC	E
CB04	RLC	H
CB05	RLC	L
07	RLCA	
ED6F	RLD	
CB1E	RR	(HL)
DDCB051E	RR	(IX+d)
FDCB051E	RR	(IY+d)
CB1F	RR	A
CB18	RR	B
CB19	RR	C
CB1A	RR	D
CB1B	RR	E
CB1C	RR	H
CB1D	RR	L
1F	RRA	
CB0E	RRC	(HL)
DDCB050E	RRC	(IX+d)
FDCB050E	RRC	(IY+d)
CB0F	RRC	A
CB08	RRC	B
CB09	RRC	C
CB0A	RRC	D
CB0B	RRC	E
CB0C	RRC	H
CB0D	RRC	L
0F	RRCA	
ED67	RRD	
C7	RST	00H
CF	RST	08H
D7	RST	10H
DF	RST	18H
E7	RST	20H
EF	RST	28H
F7	RST	30H
FF	RST	38H
DE20	SBC	A,n

Código Objeto	Instrução	
9E	SBC	A,(HL)
DD9E05	SBC	A,(IX+d)
FD9E05	SBC	A,(IY+d)
9F	SBC	A,A
98	SBC	A,B
99	SBC	A,C
9A	SBC	A,D
9B	SBC	A,E
9C	SBC	A,H
9D	SBC	A,L
ED42	SBC	HL,BC
ED52	SBC	HL,DE
ED62	SBC	HL,HL
ED72	SBC	HL,SP
37	SCF	
CBC6	SET	0,(HL)
DDCBC5C6	SET	0,(IX+d)
FDCB05C6	SET	0,(IY+d)
CBC7	SET	0,A
CBC0	SET	0,B
CBC1	SET	0,C
CBC2	SET	0,D
CBC3	SET	0,E
CBC4	SET	0,H
CBC5	SET	0,L
CBCE	SET	1,(HL)
DDCB05CE	SET	1,(IX+d)
FDCB05CE	SET	1,(IY+d)
CBCF	SET	1,A
CBC8	SET	1,B
CBC9	SET	1,C
CBCA	SET	1,D
CBCB	SET	1,E
CBCC	SET	1,H
CBCD	SET	1,L
CBD6	SET	2,(HL)
DDCB05D6	SET	2,(IX+d)
FDCB05D6	SET	2,(IY+d)
CBD7	SET	2,A
CBDC	SET	2,B
CBD1	SET	2,C
CBD2	SET	2,D
CBD3	SET	2,E
CBD4	SET	2,H
CBD5	SET	2,L
CBD8	SET	3,B
CBDE	SET	3,(HL)
DDCB05DE	SET	3,(IX+d)
FDCB05DE	SET	3,(IY+d)
CBDF	SET	3,A
CBD9	SET	3,C
CBDA	SET	3,D
CBDB	SET	3,E
CBDC	SET	3,H
CBDD	SET	3,L
CBE6	SET	4,(HL)
DDCB05E6	SET	4,(IX+d)
FDCB05E6	SET	4,(IY+d)
CBE7	SET	4,A
CBE0	SET	4,B
CBE1	SET	4,C
CBE2	SET	4,D
CBE3	SET	4,E
CBE4	SET	4,H
CBE5	SET	4,L
CBEE	SET	5,(HL)
DDCB05EE	SET	5,(IX+d)
FDCB05EE	SET	5,(IY+d)
CBEF	SET	5,A
CBE8	SET	5,B
CBE9	SET	5,C
CBEA	SET	5,D
CBEB	SET	5,E
CBEC	SET	5,H
CBED	SET	5,L
CBF6	SET	6,(HL)
DDCB05F6	SET	6,(IX+d)
FDCB05F6	SET	6,(IY+d)
CBF7	SET	6,A
CBF0	SET	6,B
CBF1	SET	6,C
CBF2	SET	6,D
CBF3	SET	6,E
CBF4	SET	6,H
CBF5	SET	6,L
CBFE	SET	7,(HL)
DDCB05FE	SET	7,(IX+d)
FDCB05FE	SET	7,(IY+d)
CBFF	SET	7,A
CBF8	SET	7,B
CBF9	SET	7,C
CBFA	SET	7,D
CBFB	SET	7,E
CBFC	SET	7,H
CBFD	SET	7,L
CB26	SLA	(HL)
DDCB0526	SLA	(IX+d)
FDCB0526	SLA	(IY+d)
CB27	SLA	A
CB20	SLA	B
CB21	SLA	C
CB22	SLA	D
CB23	SLA	E
CB24	SLA	H
CB25	SLA	L
CB2E	SRA	(HL)
DDCB052E	SRA	(IX+d)
FDCB052E	SRA	(IY+d)
CB2F	SRA	A
CB28	SRA	B
CB29	SRA	C
CB2A	SRA	D

Código Objeto	Instrução	
CB2B	SRA	E
CB2C	SRA	H
CB2D	SRA	L
CB3E	SRL	(HL)
DDCB053E	SRL	(IX+d)
FDCB053E	SRL	(IY+d)
CB3F	SRL	A
CB38	SRL	B
CB39	SRL	C
CB3A	SRL	D
CB3B	SRL	E
CB3C	SRL	H
CB3D	SRL	L
96	SUB	(HL)
DD9605	SUB	(IX+d)
FD9605	SUB	(IY+d)
97	SUB	A
90	SUB	B
91	SUB	C
92	SUB	D
93	SUB	E
94	SUB	H
95	SUB	L
D620	SUB	n
AE	XOR	(HL)
DDAE05	XOR	(IX+d)
FDAE05	XOR	(IY+d)
AF	XOR	A
A8	XOR	B
A9	XOR	C
AA	XOR	D
AB	XOR	E
AC	XOR	H
AD	XOR	L
EE20	XOR	n

(Com a amável autorização da Zilog Inc.)

APÊNDICE E TABELA DE CONVERSÃO HEXADEXIMAL

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	00	000
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	256	4096
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	512	8192
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	768	12288
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	1024	16384
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	1280	20480
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	1536	24576
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	1792	28672
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	2048	32768
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	2304	36864
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	2560	40960
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	2816	45056
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	3072	49152
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	3328	53248
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	3584	57344
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	3840	61440

5		4		3		2		1		0	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0	0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15

Outros titulos publicados pela Editora Manole :

— BASIC Basico
David Munro

— Voce Conhece os Computadores ?
Karen Billings

— Entendendo o Computadores
Myles Walsh

— MSX Jogos em Assembler
Eric Savis

— MSX Jogos de Acao
Pierre Monsaut

— MSX Rotinas Graficas em Assembler
Steve Webb

— MSX Programas em Linguagem de Maquina
Steve Webb

— MSX Tecnicas de Programacao de Jogos em Assembler
Fagot-Barraly

— Mais Jogos e Graficos para o MSX
Graham Carter

— SPECTRUM Jogos de Acao
Pierre Monsaut

— Jogos de Acao para o APPLE Faca seus proprios jogos
Zimmerman&Zimmerman

— Aulas Praticas de BASIC para o Apple Volume 1 *Hubert Alqueres*

— 250 Regras de Boa Programacao
Ledin & Ledin

Caso nao encontrar o livros na sua livraria, procurar
na editora que fica na Rua 13 de Maio , 1026 – Bela Vista – SP
(travessa da Av. Brig. Luis Antonio)
Fone: 287-0746

Impresso nas oficinas da
EDITORA PARMA LTDA.
Fone: 209-5077
Av. Antônio Bardella, 280
Guarulhos - São Paulo - Brasil
Com filmes fornecidos pelo Editor

Todos os usuários de microcomputadores MSX, após terem dominado os recursos do BASIC, desejosos de progredir encontrarão nesta obra os elementos necessários para abordar a progamação em linguaguem de máquina ou em assembler.

Após rápida recordação de aritmética binária, as principais instruções do microprocessador Z80 são descritas e acompanhadas de exemplos de subprogramas em código de máquina integrados a um programa BASIC. O leitor aprenderá como realizar programas muito mais complexos compreendendo também melhor o funcionamento do seu microcomputador.

O AUTOR

Georges Fagot-Barraly é professor de matemática e especialista na utilização de microcomputadores no ensino. Ele realizou numerosos programas didáticos e é também autor de uma série de obras sobre programação em assembler.

MSX